TREBALL FINAL DE GRAU

TÍTOL DEL TFG:	**Disseny i construcció d'una sonda atmosfèrica**
TITULACIÓ:	**Grau en Enginyeria d'Aeronavegació**
AUTORS:	**Castellà Ayala, David** **Fournier Cairó, Yannick**
DIRECTOR:	**Gutiérrez Cabello, Jordi**
CODIRECTORA:	**Gil Pons, Pilar**
DATA:	**28 de Novembre del 2016**

Resum

El projecte consisteix en la construcció d'una sonda atmosfèrica controlada per un microcontrolador Arduino. Aquesta sonda ha de poder obtenir dades de pressió i temperatura en funció de l'alçada. La construcció d'aquest dispositiu, el primer d'aquest tipus a l'EETAC, s'ha realitzat partint des de zero, sent necessària doncs una àmplia tasca de definició de requisits. En la present memòria descrivim com s'han dissenyat tots els subsistemes que formen part de la sonda, enumerant tots els components necessaris així com el motiu per el qual s'han escollit. El disseny resultant és modular per facilitar futures millores/expansions.

També s'han detallat les passes necessàries per a l'assemblatge de tots els components en una estructura comuna, així com la elecció de eines i materials. S'han posat a prova tots els sistemes creats, comprovant si els valors obtinguts compleixen els requisits definits.

Per últim, una vegada el dispositiu ja s'ha construït i validat, s'han detallat totes les tasques que cal realitzar per sol·licitar el llançament de la sonda a una alçada de 35km, així com els materials necessaris. S'ha plantejat aquesta memòria tècnica amb la funcionalitat addicional servir com a manual. El dispositiu resultant s'ha batejat com a FourCast.

TITLE:	**Design and construction of a meteorological probe**
DEGREE:	**Grau en Enginyeria d'Aeronavegació**
AUTHORS:	**Castellà Ayala, David** **Fournier Cairó, Yannick**
DIRECTOR:	**Gutiérrez Cabello, Jordi**
CODIRECTOR:	**Gil Pons, Pilar**
DATE:	**November 28th 2016**

Overview

The project consists of the construction of a functional meteorological probe, controlled by an Arduino microcontroller. This probe was design to measure pressure and temperature as functions of the altitude. This device is the first of its kind built at the EETAC, thus a considerable effort of requirement definition has been done. In the present report we describe how all the probe systems were designed, and all the necessary components as well as the reason why they were chosen are described. The resulting design is modular in order to facilitate future improvements/expansions.

The steps necessary for the assembly of all the components in a common structure are detailed, as well as the choice of tools and materials. All the systems developed were tested simulating conditions similar to those expected in the real mission.

Finally, after the construction and validation processes, all the materials and the tasks needed to launch the probe up to at an altitude of 35 km are detailed. The present report is intended to serve as a guide for future similar projects in EETAC. The resulting device from this work is named FourCast after our surnames.

To my family, specially my mother
and to all the friends that supported
me during this long journey

To my friends and family, whose
unconditional support helped me
in the darkest hours

To Pilar, Jordi, Oscar and Joan for them
guidance and knowledge, as well as
all the external agents that helped us,
such as ACTIS and Escola Jacint Verdaguer.

And thanks also to the support of the EETAC,
which made feasible the implementation
of this project

Index

List of Figures

List of Tables

Glossary

AAI: American Astronautical Society
ABS: Acrylonitrile Butadiene Styrene
ADC: Analog Digital Converter
ARLISS: A Rocket Launch for International Student Satellites
AIAA: American Institute of Aeronautics and Astronautics
BNC: Bayonet Neil-Concelman
CMOS: Complementary Metal-Oxide-Semiconductor
CNES: Centre National d'Études Spatiales
DAC: Digital Analog Converter
DHCP: Dynamic Host Configuration Protocol
DITESA: Design and Test of Aeronautical Systems
DNS: Domain Name System
EEPROM: Electrically Erasable Programmable Read-Only Memory
EETAC: Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldedels
ESA: European Space Agency
FM: Fading Margin
GND: Ground
GNSS: Global Navigation Satellite System
GPS: Global Positioning System
HSDPA: High-Speed Downlink Packet Access
HTTP: Hypertext Transfer Protocol
IoT: Internet of Things
ISA: International Standard Atmosphere
LEEM: Laboratory for Space and Microgravity Research
NACC: Navegació Aèria, Cartografia I Cosmografia
NOTAM: Notice to Airmen
OPAMP: Operational Amplifier
PC: Personal Computer
PCB: Printed Circuit Borad
PLA: Polyactic Acid
PWC: Pulse Width Count
RESTful: Representational State Transfer
RTD: Resistance Temperature Detector
RTS: Request To Send
SaTReC: Satellite Technology Research Center
SMA: SubMiniature version A
SQL: Structured Query Language
TCP: Transmission Control Protocol
UDP: User Datagram Protocol
UNISEC: University Space Engineering Consortium
UPC: Polytechnical University of Catalonia
UPM: Polytechnical University of Madrid
USB: Universal Serial Bus
WCF: Windows Communication Foundation

Introduction

State of the art

In November 1998 US and Japanese universities jointly organized the University Space Systems Symposium [1]. New exciting concepts were presented by Prof. Bob Twiggs, from Stanford University: CanSats and CubeSats were introduced as low-cost artificial satellites which could be afforded by educational institutions and small companies, rather than only by space agencies. A set of new small satellite requirements was established in this meeting. It was the start of a new generation of small satellites, which took off with the first launch only one year later, on September 11th in Nevada, in the Black Rock desert, with the ARLISS (A Rocket Launch for International Student Satellites) program [2].

This first launch was performed with a rocket, which brought a CanSat up to almost 4000 m height and then released it. A series of different launches were executed afterwards with different objectives [3]. Since then many different improvements were conducted, such as the implementation of a GPS in 2000 or the beginning of a new type of CanSat in 2001, which would be able to come back to a specific location. Furthermore, the program kept growing and growing and it went off the borders of the US territory. Nowadays, many countries hold competitions aimed to launch the best CanSats designed and constructed by students. In Japan, the CanSat competitions are organized by the UNISEC (University Space Engineering Consortium) and instead of launching their probes with a rocket they use balloons to carry them to the desired height.

Nowadays National Space Agencies and leading space research centers are really taking an interest in the high potential of CanSat projects in terms of offering creativity, capacity of innovation and recognition of new talents. In France, the CNES (Centre National d'Études Spatiales) is responsible for the CanSat competition. In South Korea, SaTReC (Satellite Technology Research Center) is the organizer of the equivalent contest, which has the sponsorship of the Korean Ministry of Science. In Spain, there are two main organizers; the Laboratory for Space and Microgravity Research (LEEM) and the Polytechnic University of Madrid (UPM), which held in 2008 an International CanSat competition. From a European point of view, the organizer is the European Space Agency (ESA) and in the United States both the American Astronautical Society (AAI) and the American Institute of Aeronautics and Astronautics (AIAA) are responsible of the CanSat competition.

From the point of view of the design a CanSat must fit in the volume of a regular soda can (350 ml) and have a maximum mass of 500 g. There are two main parts that are compulsory in a CanSat design. The first one is the power supply, which must be given by a battery. This battery can be either rechargeable or non-rechargeable. The second element is the microcontroller. It is necessary to have a small computer on-board to allow all the secondary systems to operate.

Once these two elements, which are the base of a CanSat, have been implemented, there are different components which can be added, depending on the technological or scientific goals to be achieved. These additional components

are called the secondary elements. The most common secondary elements are a temperature system and a pressure measurements systems. Other equipment could consist of a GPS to track the position of the CanSat, a camera, or a Geiger counter to measure ionizing radiation [4].

According to the standard classification of ESA [5], depending on the mission design, there would be different types of CanSat. The telemetry CanSat is aimed to data acquisition. It takes measurements of temperature, pressure, GPS positioning or even radiation levels. There is also the “Comeback” CanSat, which returns to a specific location programmed before the mission.

From the point of view of the launch a CanSat can be placed inside a rocket, or in a helium balloon that elevates it to the desired altitude.

Motivation of this project

Since the idea was introduced in the late 1990s, picosatellite design and construction has made space research available with low budget, and thus widely accessible for education purposes. Students from all around the world can join small space projects and get practical experience of the ideas which are learnt in classrooms. This opens a new way to explore and learn key concepts in the engineering world.

From the personal point of view, when we decided to do this project we knew it would be a huge challenge for us, however we were confident about carrying it away. There were many things that motivated us to accept this challenge. Firstly, and most importantly, we wanted to do this project because we liked the idea and were ready to invest time doing it. We were also told that this project would be the first one of its genre done by the EETAC. This was also a challenge for us because there were high expectations set on us. We knew that if this project would be well sorted out, other students would try and find new solutions or new ways of designing and building a CanSat-like probe in order to improve our prototype and could be a great asset for the school.

Another key aspect that helped us to take this decision was the wide range of courses that played an important role in this project. Thanks to the knowledge and concepts of other courses we were able to have a better understanding of some aspects of our device. Courses like Avionics or Aeronautical Communications were important to design the whole electronic system (pressure, temperature, Arduino, etc…) and build the budget link of our device. The subject DITESA (Design and Test of Aeronautical Systems) also helped us to design the power supply system and deciding which will be the better battery for our system. The following subjects also had an important role to design our device: Radiolocation, Communications Fundamentals and NACC.

Structure of the project

In the present work, we consider the CanSat philosophy in order to design and build a low-cost experiment. As we will explain along this report, an actual CanSat could not be built because of budget restrictions, as miniaturized components were more expensive than the standard ones we have used. Nevertheless, we present a fully operational system and leave miniaturization options for future works. This project is structured in five chapters.

Chapter 1 is the most extensive. This chapters explains the different systems that form the probe. We define which requirements do we need for each one of the systems, and justify the selection of the chosen electronical components. We will also explain and justify the electronical design of the systems and include test results from simulations and Protoboard tests.

Chapter 2 explains how the individual systems described in chapter 1 can be fitted together into a common structure, also remarking the special requirements of that structure, including materials and PCB assembly. This chapter also includes full measures of the structure we built for the FourCast.

Chapter 3 explains the validations that were conducted once the device was assembled. It enumerates the design of the different tests, and the validation results for each system. It also explains the different actions that were taken when validations did not succeed.

Chapter 4 explains all the work did in order to prepare the launch of the probe. It enumerates the materials needed in order to make a successful experiment, and also the administrative work needed to perform the launch.

Chapter 5 contains the conclusions of the project, and also possible improvements as well as our expectations for the future and continuity of this kind of projects at EETAC.

After the main content of the project, we included 8 annexes that intend to make our work more comprehensive and add additional information.

Annex A includes the technical datasheets, Annex B includes the source code of the programs we made, Annex C includes detailed test results complementary for chapter 3, Annex D includes the launch checklist, Annex E the NOTAM requests needed to perform the experiment, Annex F all the planning tools that we used during the making of the project, Annex G includes a list of all materials that we use in the building of the FourCast experiment, and also the price and final cost of the project. Finally, Annex H includes a list of EETAC courses related to this project.

CHAPTER 1. ELECTRONIC DESIGNS OF FOURCAST SUBSYSTEMS

In this chapter, we describe and justify our design choices for the different subsystems of FourCast, the probe we have built. Keeping in mind the CanSat philosophy the power supply is given by batteries and a microcontroller communicates with the different subsystems and, when required, converts incoming analog signals to digital.

The entire design process is determined by the requirements of the probe. In our case, we want FourCast to be able to take measurements of atmospheric temperature and pressure, and be able to send this information to a ground station. Besides we intend the device to be recoverable.

We have designed FourCast to be very modular, in the sense that subsystems must be able to work independently, except for the microcontroller, which acts as a common bridge between them.

Some of the systems are vital for the mission, as the sensors and the power supply, other add enhanced features that make FourCast more reliable and useful, for example the recovery system. Each system has been carefully designed, choosing the proper components and making all the electronic circuitry needed to make it work. Figure 1.1 summarizes all of the integrated systems.

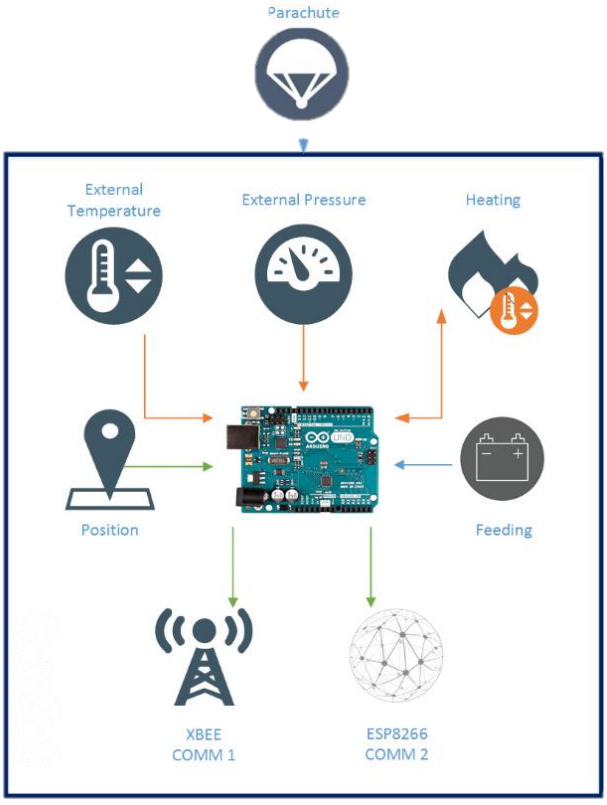


Figure 1.1 FourCast systems. Orange indicates analog and green digital.

1.1 FourCast external temperature sensor

The external temperature system is one of the most important systems of this experiment, as the objective of this mission is to obtain a precise reading of the temperature outside the FourCast during its ascent. Given the nature of the experiment and due to the high altitude, that will be achieved by the FourCast a wide range of temperatures is expected. More precisely, the system should be able to provide accurate readings for temperatures between -65 and 35 degrees Celsius, a range of 101 degrees.

With such a wide range the external temperature system must have an appropriate sensor, capable of operating at rather low temperatures and a conditioning system that provides enough accuracy for the whole range. A Platinum RTD sensor seems to be an appropriate choice, as it works like a variable resistor decreasing and increasing its impedance almost linearly with the temperature, following the equation (for a Pt-1000 RTD):

$$R = 1000 \cdot (1 + 0.00385 \cdot \Delta T) \quad [\Omega] \qquad \textbf{(1.1)}$$

where ΔT is the temperature in degrees Celsius. In order to achieve this, we have chosen a Platinum RTD that offers 1000Ω resistance at 0˚C. More specifically the sensor chosen is the IST DIN-EN-6075 [6] (view Annex A), with the packaging 6W 232 which fulfills all our needs.

The Platinum RTD must send the probe's processor (Arduino UNO) a range of voltages that can be translated back into temperature. Given the nature of the RTD sensor the easiest way to achieve this goal is with a voltage divider. After that the output of the voltage divider must be adjusted in order to fulfill the ADC range of the Arduino board and take maximum advantage of the 10-bit quantifier. The components we need for the RTD input stage are the following:

- 1x Pt-1000 RTD temperature sensor.
- 1x 10kΩ 5% ¼W resistor.
- 1x 100uF 20% capacitor.

The components needed for the amplifier stage are the following:

- 1x LM-324N Quad Operational Amplifier
- 1x 100kΩ 5% ¼W resistor.
- 1x 10kΩ potentiometer.
- 2x 10kΩ 5% ¼W resistor.
- 2x 68kΩ 5% ¼W resistor.
- 1x 220kΩ 5% ¼W resistor.

1.1.1 Design process

The design of the external temperature system has been performed taking into account the following conditions:

- Elevated currents will provoke a high self-heating error on the RTD, so the voltage that falls at the RTD should be low. In our case, we also benefit from the fact that we will be reading mostly low temperatures, so the resistance value of the RTD will be lower than in other cases.
- High value resistors will increase the overall noise, due their tolerances.
- High value resistors will also decrease the power consumption of the system, increasing the duration of the battery.

Taking all this into account we have created a design that balances the requirements stated above.

1.1.1.1 RTD Input Stage

In order to obtain a voltage as an output from the RTD a voltage divider will be used, using a 10kΩ resistor. This value may seem rather high, but this will ensure that low voltages will fall on the RTD (see Figure 1.2).

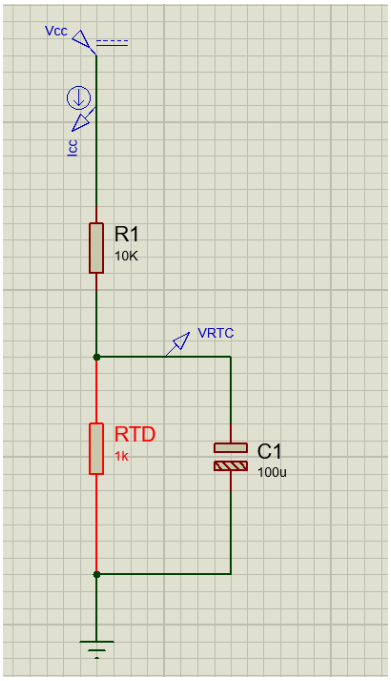


Figure 1.2 Simulation of the voltage that falls at the RTD

Knowing the values that the RTD can have for our range of temperatures we can also know the values of V_{RTD}.

$$R_{35} = 1000 \cdot (1 + 0.00385 \cdot 35) = \quad 1134.75\Omega \qquad \textbf{(1.2)}$$

$$R_{-65} = 1000 \cdot (1 - 0.00385 \cdot 65) = \quad 749.75\Omega \qquad \textbf{(1.3)}$$

And knowing that the output voltage out of a voltage divider is:

$$V_{RTD} = V_{cc} \cdot \frac{RTD}{RTD + R1} \qquad \textbf{(1.4)}$$

Then, in our case, considering a temperature range between -65 and 35 degrees Celsius the output voltages will be:

$$V_{RTD\ -65} = 5 \cdot \frac{749.75}{749.75 + 9800} = 0.3553V \qquad \textbf{(1.5)}$$

$$V_{RTD\ 35} = 5 \cdot \frac{1134.75}{1134.75 + 9800} = 0.5189V \qquad \textbf{(1.6)}$$

This values are low enough to assure a low self-heating error on the RTD measurements, as the associated low heating power can be easily dissipated. The capacitor is added in order to serve as a low-pass filter. This is important in order to eliminate noise in high frequencies, which appear to be irrelevant because temperature varies slowly in our case.

The 10-bit ADC of the Arduino UNO board can use two reference voltages for the quantification without need of external components: 5V or 1.1V. Taking into account the values of V_{RTD}, 1.1 V might *a priori* seem a reasonable value but it is not actually sufficient. The 10-bit ADC divides the 1.1V into 1023 steps of 1.07mV. Our V_{RTD} range varies 163.6mV. This means that only 152 steps out of the 1023 possible would be used, only 15% of the full range of the quantifier. As a 100˚C temperature range must be read quantification errors of $\pm 0.66^{\circ}C$ would be expected. It is a value rather high, which may even get higher if we take into account that there will also be additional sources of errors.

Therefore, we will use an operational amplifier in order to obtain a voltage output of 0V when the temperature is -65˚C and a voltage of 1.1V when the temperature is 35˚C, covering the sensor's full range and the full range of the Arduino 10-bit ADC. We will use a differential amplifier to achieve this goal.

1.1.1.2 Amplifier Stage

We have chosen the model LM324N [7] (see Figure 1.3) (view Annex A) as it is a low-power and low-noise amplifier. This is rather good for the energy consumption and also for the accuracy of the readings.

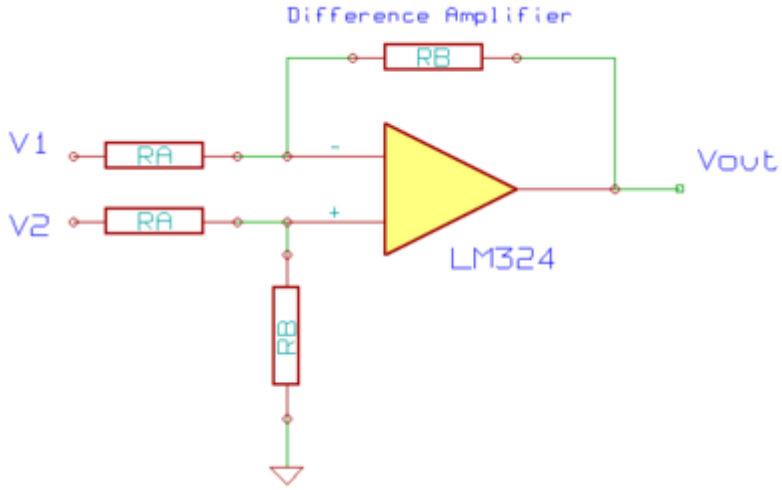


Figure 1.3 Differential Amplifier

With the previous setup, the voltage out of the amplifier can be expressed as:

$$V_{out} = \frac{R_B}{R_A} \cdot (V_2 - V_1) \qquad \textbf{(1.7)}$$

In our case V_2 will be $V_{RTD,}$ the output of the RTD voltage divider.

So, if we want to have 0V when the temperature is -65˚C we will need that both V_1 and V_2 are 0.3553V. In other words, we need to set V_1 at the same voltage as the RTD output at -65˚C. This can be done using another voltage divider, with two fixed resistors that will always have an output voltage of 0.3487V. In our case, we have chosen a 100kΩ resistor and a 10kΩ potentiometer that we will adjust until we obtain a fixed output voltage of 0.3553V. The resistor value of the potentiometer should be, approximately:

$$0.3553 = 5 \cdot \frac{RPOT}{RPOT + 100000} \rightarrow RPOT = 7649.57\Omega \qquad \textbf{(1.8)}$$

The LM324N chip includes 4 operational amplifiers, so we will use one as a voltage follower, in order to prevent the possibility that the operational amplifier might load the voltage divider in a way that could alter the fixed output voltage (see Figure 1.4).

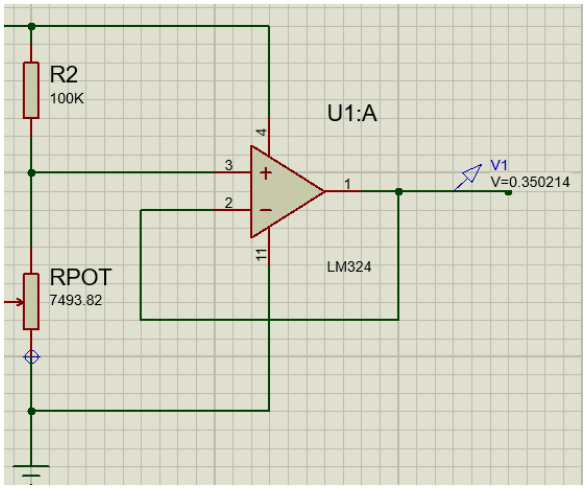


Figure 1.4 Proteus simulation of the voltage divider for V_1

At 35˚C the value of V_{RTD} will be 0.5189V. This means that $V_{RTD} - V_1$ will be 0.1636V. We need to amplify this to 1.1V, so we need a linear gain of 6.7; we need the ratio $\frac{R_B}{R_A} = 6.7$ in the differential amplifier.

Then R_A will be equal to 10 kΩ and $R_B$will be equal to 67 kΩ. The ratio gives us 6.7, as desired and also the value of the resistors are high enough to assure low power consumption but not high enough to cause a high level of noise. This amplification of the signal allows to use the full range of the ADC, and therefore achieve a reasonable resolution:

$$\frac{100℃}{1023\ steps} \approx 0.1℃ \qquad \textbf{(1.9)}$$

Which is enough for our requirements.

1.1.2 Input to Arduino UNO Board

The output of the difference amplifier will be the input of an Analog Read port of the Arduino board, but between the output of the difference amplifier and the Arduino input a resistor of 220kΩ will be added. This is done in order not to saturate the Arduino input port.

1.1.3 FourCast external temperature sensor complete system

The full design of the external temperature reading system has been simulated in Proteus using real components. The obtained values satisfy the needs of the project (see Figure 1.5).

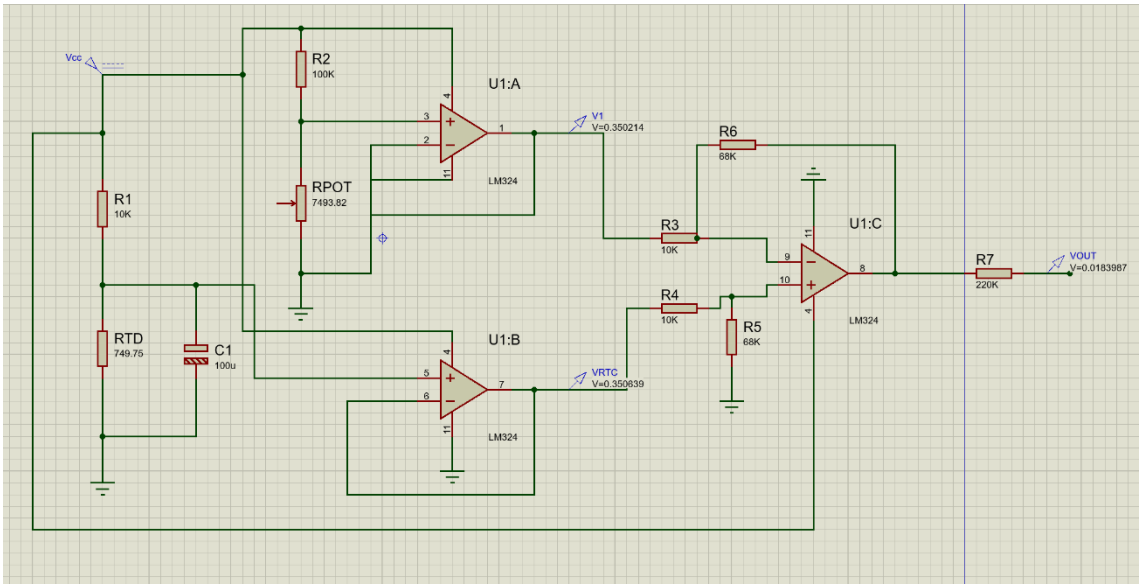


Figure 1.5 Complete external temperature reading system

VOUT will be introduced into an analog port of the Arduino. The obtained values of the simulation are shown in Table 2.

Temperature (˚C)	VOUT (V)
-65	0.0184
0	0.7378
35	1.1117

Table 1.1 Results of the simulation

As it is a linear sensor, the processing on the Arduino board will be simple. In order to reduce random noise, we take 15 samples equally separated in a total of 3 seconds. Then, the temperature can be obtained simply as:

$$External\ Temperature = \frac{V_{out}\ (mV)}{11(mV/°C)} \qquad \textbf{(1.10)}$$

The full code of the system can be viewed at Annex B.

1.2 FourCast external pressure sensor

One of the most interesting capabilities of the FourCast will be the ability to obtain precise pressure measurements. The pressure can be used for multiple purposes. We can use its values to calculate altitude, like in a standard altimeter. This measurement can complement the GPS readings, and offer us an accurate view of the precision of the International Standard Atmosphere (ISA) tabular data model, which was last updated in 1976. Choosing the right measurement equipment is a key decision in the whole probe project.

Pressure sensors can be used in multiple situations. Basically, there are two type of sensors: absolute and differential.

- Absolute pressure sensors (see Figure 1.6) normally have only one input tube, and they read the absolute pressure, meaning the local pressure plus an offset, typically the atmospheric pressure. This kind of sensor is used, for example, in altimeters.

- Differential pressure sensors (see Figure 1.7) normally have two input tubes. They read pressure from the two tubes and the output is the difference between them. This is useful when the user is only interested in local pressure values, because one of the tubes may remain “open” and then the atmospheric pressure is subtracted. This kind of sensor is used, for example, in manometers.

Figure 1.6 Absolute pressure sensor.

Figure 1.7 Differential pressure sensor.

As our goal is to measure atmospheric pressure, so, the convenient choice is an absolute pressure sensor. We also have to consider the range of expected pressures. If we want to achieve a 35km altitude we have to expect very low pressures, according to ISA around 1kPa. Then, the range of our sensor has to be prepared for pressure measurement from sea level values(101.3kPa) and down to pressures at 35km (5kPa). This will be a key factor to decide the appropriate sensor. Given these parameters the chosen sensor has been the MPX2200AP of Freescale (Motorola) [8], shown in Figure 1.8 (view Annex A).

Figure 1.8 MPX2200AP case.

The sensor behaves like a Wheatstone bridge of 4 resistors. It gives the output as the difference of two output voltages. Sensor's range varies between 0 and 200KPa, covering the expected interval of FourCast’s pressure measurements.

According to the manufacturer the output voltage at 0KPa is 0mV and at 200kPa is 40mV, being linear in the whole range and with a sensitivity of 0.2mV/kPa. This range of voltages is too low to be used directly as an input for the Arduino UNO ADC, as we would get a resolution of 24kPa (note that 40mV represents 0,8% of the ADC range and the precision is 10 bit). This is unacceptable for our purposes and we need to accommodate the circuit.

As we have explained, the sensor has two voltage outputs and the overall output voltage is the difference between them. This is why the accommodation is done with a differential amplifier. This particular sensor needs to be feed with 10V, which is the double of the 5V Arduino voltage output. In order to obtain 10V a CMOS Monolithic Voltage Converter was used, which is capable to act as a voltage inverter or voltage doubler. In our case, we used a MAX660 Converter [9] (view Annex A). For the MPX2200AP it is highly recommended to create the 10V using positive and negative 5V. If the sensor is referred to 10V and GND, the output voltages that contain the pressure information are highly unstable. The components used are:

- 1x MPX2200AP Absolute Pressure Sensor.
- 2x 10kΩ 5% ¼W resistor.
- 2x 470 kΩ 5% ¼W resistor.
- 2x 10uF 20% capacitor.
- 1x LM-324N Quad Operational Amplifier.
- 1x MAX660 CMOS Monolithic Voltage Converter

1.2.2 Design process

The design of the pressure system will be divided in two blocks, one for the subsystem that feeds the sensor and another one for the processing of the data itself, and accommodation to the ADC range.

1.2.2.1 Sensor power supply using a voltage inverter

The MAX660 is a small all-integrated chip that only needs two external capacitors in order to supply a stable negative voltage value, given a positive voltage input. It can also work as a voltage doubler, but this function is not suitable for our needs as the output voltage from the pressure sensor is then too high and the LM324N amplifier saturates. The connections for this purpose have to be made following Figure 1.9 schema.

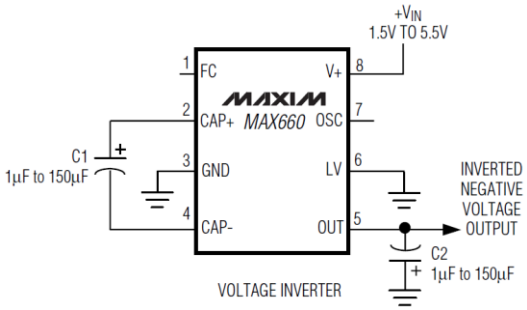


Figure 1.9 Schematic for the MAX660 as a Voltage inverter.

Then, the PIN1 of the sensor must be connected to the -5V and the PIN3 to the positive 5V given by the Arduino.

1.2.2.2 Data acquisition

The main objective of the conditioning system is to adjust the range of output voltages to the range of the Arduino board ADC. This has to be done taking into account that we want to minimize the power consumption and also minimize the generated noise. The voltage reference of the ADC will be set again to 1.1V.

We will adjust the components of the differential amplifier in order to obtain 1.1V at sea level approximately, which is the maximum value of pressure expected. Luckily, the output voltage at 0kPa is also 0V, so no offset needs to be subtracted. The sensor output at sea level is 20mV, so the tension needs to be amplified by a factor of 55 (1.1/0.020). According to the expression for the differential amplifier:

$$V_{out} = \frac{R_B}{R_A} \cdot (V_2 - V_1) \quad \textbf{(1.11)}$$

$$R_B = 470k\Omega \quad \textbf{(1.12)}$$

$$R_A = 10k\Omega \quad \textbf{(1.13)}$$

The gain factor is 47 in this case, because it is easy to find 470kΩ resistors. Therefore, the expected values at the output of the amplifier will be 9.28mV/kPa.

$$\frac{0.020\ V * 47}{101.3\ KPa} = 9.28\ mV/kPa \quad \textbf{(1.14)}$$

The range is approximately from 0 to 118.5kPa, and given the ADC resolution, the overall resolution of the output will be around 0.115kPa.

$$\frac{1100\ mV}{9.28\ mV/KPa} = 118.5\ kPa \quad \textbf{(1.15)}$$

$$\frac{118.5\ KPa}{1023} = 0.115\ kPa \quad \textbf{(1.16)}$$

The positive output of the sensor must be connected to the + pin of the differential operational amplifier, and the negative one to the – pin. No voltage followers were used this time as there is no load effect expected. Then we could use a single Quad OPAMP LM324N chip to accommodate both pressure and external temperature systems as in Figure 1.10.

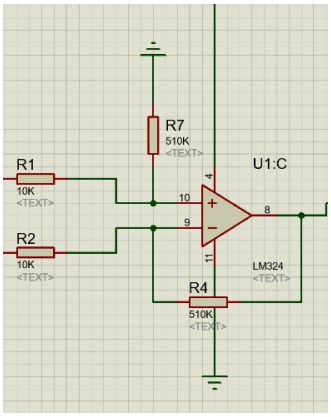


Figure 1.10 Differential amplifier for the Pressure Sensor System.

The output of the differential amplifier will be introduced into an analog port of the Arduino. The obtained values of the simulation are presented in Table 3.

Pressure (kPA)	Output (V)
0	0
50	0.464
100	0.928

Table 1.2 Results of the simulation

As it is a linear sensor, the processing on the Arduino board will be simple. In order to reduce random noise, we take 15 samples equally separated in a total of 3 seconds. Then, the pressure can be obtained simply as:

$$External\ Pressure = \frac{V_{out}\ (mV)}{9.28\ (mV/kPa)} \quad \textbf{(1.17)}$$

The full code of the system can be found at Annex B.

1.3 FourCast main communications system

The communication system is critical in our entire project, due to its complexity and the essential role it plays for FourCast probe. This system obtains all the information read by the Arduino and transmits it to the ground station. All this information, which is none other than the telemetry, will be transmitted to the receiver station as soon as it is gathered by the microcontroller. Thus, we will be able to know live all the values measured by the FourCast payload. Another option might be having an additional storage unit inside our probe so that telemetry values can be gathered after the landing. However, this second option, despite of being easier, is not useful in our case, we need to have live access to the telemetry information. This way we can infer FourCast's behavior in terms of its position and, furthermore, in case recoverability became impossible, the mission data would still be available.

In order to design this system, we need to have the following products:

- 2 XBee PRO 868MHz
- 0.6 dB antenna for Tx
- 17 dB YAGI antenna for Rx
- Arduino XBee Shield
- USB Shield PC to XBee
- BNC – SMA adapter
- SMA male – SMA female adapter.

Our choice for the transmitter (Tx and Rx) was the XBee PRO 868MHz [10] (view Annex A), which is the best option we have found. It is small, designed to operate with the Arduino, it has a long RF line-of-sight range (of more than 40km according to the manufacturer), its transmission power is 315mW, the receiver sensitivity is -112 dBm and it works at 3.3V. This should not be an inconvenient because the Arduino can provide this voltage. In addition, the best way to connect the XBee to the Arduino is a shield [11]; this one will be directly connected to the microcontroller so that it can continue to facilitate the connection to its pines. The XBee was placed on top of the shield. The XBee has a SMA connector, so the antenna connected to this XBee must also have an SMA connector, and besides it should be a low-gain antenna with low directivity (see Figure 1.11), as it is difficult to get continuously high precision information about the FourCast's inclination and attitude. As a consequence, we chose to use a monopole antenna. [12] (view Annex A).

Figure 1.11 XBees (Tx & Rx), Shields (Arduino & USB) and antenna.

This system does not need any additional wire connecting the microcontroller to the XBee because the shield does all these functions, already allowing the appropriate communications between both components. There is a switch on top of the shield which allows the user to select between "USB" and "XBee". By selecting "XBee", the XBee will send all the data printed in the serial monitor; if, in the other hand, "USB" is selected the microcontroller will send the serial monitor data through the USB port, as usual.

The XBee is capable of transmitting the serial monitor output of the Arduino, directly using the connections of the shield so all pin remains unused. All the data sent through the hardware serial ports of Arduino (0 and 1 digital) may also be sent.

Once the transmission part is completed we must consider the reception and how the communication between both XBee's might be achieved. The ground station does not need a microcontroller because a standard PC can be used. The XBee used for reception will be the same as the one used for transmission, whereas the shield in this case will be a different device [13], see Figure 1.11. As we do not have an Arduino for reception, the shield needed must have a USB connector to the PC, and the XBee will go on top of it. The antenna is also being different from the one we used for transmission. For reception, it is more convenient to use a high gain antenna. We have taken a standard YAGI antenna [14] (see Figure 1.12), used for TV, which has a BNC connector (view Annex A). We then need an adapter that goes from this BNC connector to the SMA connector we have in the XBee.

Figure 1.12 Rx antenna (17 dB gain)

The final stage to close this system relies on the communication software that we will use in order to connect both XBee's. The software that we will use is called X-CTU [15]. This software will allow us to configure both XBee's so that they have the same characteristics. After having installed the software we must connect each one of the two XBee PRO to the USB shield in order to configure them with the software.

The X-CTU software automatically detects the XBee modules connected through COM ports, and has an automatically search feature. In order to "discover" the XBee connected to the PC we must use this feature. We only need to select the COM adapter where the XBee is connected as shown in Figure 1.13 (emulated through USB).

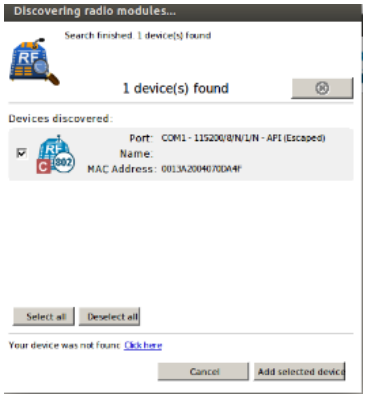


Figure 1.13 X_CTU Auto Search feature

Once the XBee device is correctly added, the user can perform multiple configuration tasks, as updating the radio firmware or changing some network parameters.

One important step of this process is to choose one of the two transmitters as the Master or Coordinator. In our case this one will be the ground station transmitter. This transmitter will be able to synchronize and communicate with other transmitters, just like when we use satellite communications, and the ground station gets information from more than one satellite. Therefore, when we connect the ground transmitter we must configure it as the *Coordinator,* and the XBee located in the probe will be called *Standard Node*.

X-CTU allows multiple of network configurations, but in our case, the simplest will be used. The radio parameters that we need to configure are the following:

- **PAN ID:** We need to establish a common network ID for both XBee. For example, 746E. This parameter is the same in both modules.

- **Destination High DH:** Here goes the High address (see Figure 1.14, DH) of the opposite XBee respect the one that we are configuring.

- **Destination Low DL:** Here goes the Low address (see Figure 1.14, DL) of the opposite XBee respect the one that we are configuring.

Figure 1.14 Back of XBee modules

- **Data Rate BD:** 9600 bauds at both.

- **Coordination Enable CE:** One XBee as Coordinator and the other one as End Device.

We will select a modulation rate of 9600 bauds, as the amount of data that we are sending is small, and we do not need high bit rates. A low bit rate also improves the energy per bit, and therefore also improves the range of the communication system. Once both transmitters are configured, we place them in

their respective positions and we tap the “Working mode” button, then we tap “Open” and we will see how both transmitters communicate.

1.3.1 Signal quality calculations

In order to ensure that the election of components was correct, we calculated the quality parameters of the received signal. This allowed us to obtain the range of communications (possible line of sight) and the quality of the signal referred to the noise. We considered different type of attenuations (L_S: free space losses, L_a: atmospherical losses and L_{IRx}: transmission line losses) and a Fading Margin (FM) of 20 dB. Additionally, we also took into account the gains of the antennas, both Rx and Tx, the sensitivity of the receiver and the transmitted power. All the values of these parameters were taken from the components datasheets.

First of all, we calculated the maximum range at which the signal can be received:

$$P_{Rx}(dBW) = P_{Tx}(dBW) + G_{Tx}(dBi) + G_{Rx}(dBi) + FM(dB) - L_s(dB) - L_a(dB) - L_l(dB) \quad \textbf{(1.18)}$$

$$L_s = 142\ dBW - 5dBW + 0.6\ dBi + 17\ dBi - 20\ dB - 3.17\ dB - 1.35\ dB = 130.08\ dB \quad \textbf{(1.19)}$$

$$130.08\ dB = \left(\frac{4\pi d}{\lambda}\right)^2 \quad \textbf{(1.20)}$$

$$d = \frac{\sqrt{1.02 \cdot 10^{13}} \cdot 0.3456m}{4\pi} = 87.77km \quad \textbf{(1.21)}$$

This range only gives us information about the electrical range of the signal, it does not inform us about the actual content of the signal, the digital data. In order to compute signal quality at data level (E_b/N_o) we need to characterize the noise figure of the receiver. We have configured the XBee modules to use a baud rate of 9600 baud/s. Assuming BPSK modulation and Nyquist criteria, we have a 9600 bits/s bit rate (R). In order to characterize the noise figure of the receiver we need to estimate the antenna temperature. Considering that the antenna is pointing to the sky, and taking into account the effect of the solar beams we estimate an antenna noise temperature (T_{ant}) of 2000K. If we use this value alongside the noise figure of the coaxial cable that joins the antenna to the receiver we obtain a global receiver noise temperature (T_s) of 2105.7K. This values allow us to compute the E_b/N_o for the budget link at our maximum distance, 35km.

$$\frac{E_b}{N_0}(dB) = P_{Tx}(dBW) + G_{Tx}(dBi) + G_{Rx}(dBi) - L_s(dB) - L_a(dB) - L_l(dB) - (K \cdot T_s \cdot R) \quad \textbf{(1.22)}$$

$$\frac{E_b}{N_0} = -5\ dBW + 0.6\ dB + 17dB - 1.35dB - 3.17dB - 122.1dB + 155.54dB = 41.5dB \quad \textbf{(1.23)}$$

With this calculation of signal quality parameters, we can assume good coverage and good quality of the transmitted data during all the experiment.

1.4 FourCast retrieval and enhanced communications system

The design of the retrieval system for the probe emerged as a necessity during the first meetings of the project. The GPS system automatically shuts down when reaching an altitude above 18.000 meters. This will be explained in detail in the GPS section of the chapter. The natural conclusion of this is accepting that the upper leg of the whole FourCast trajectory will be done under conditions of dead-reckoning navigation. The absence of accelerometers inside the structure of the FourCast makes it difficult to follow its trajectory once the GPS system shuts down. If this situation happens the telemetry collected during the lift phase can still be used, however it will not be easy to know its location when it lands because the XBee signal might have been lost.

In order to avoid this situation and to increase probability of retrieval, an additional communications system was proposed, under the following premises:

- The system must be able to communicate without depending on directive antennas.
- The system must be able to transmit the whole telemetry and parameters.
- The system must be able to work during the falling phase of the device, until landing.

Based on these requirements, the necessity of a more-complex system was clear. We decided to jump into the Internet of Things (IoT) [16] world and make the Arduino microcontroller inside the FourCast able to talk (talk IP?) IP and communicate with any device connected to Internet in the world. Figure 1.15 schematically summarizes this mission:

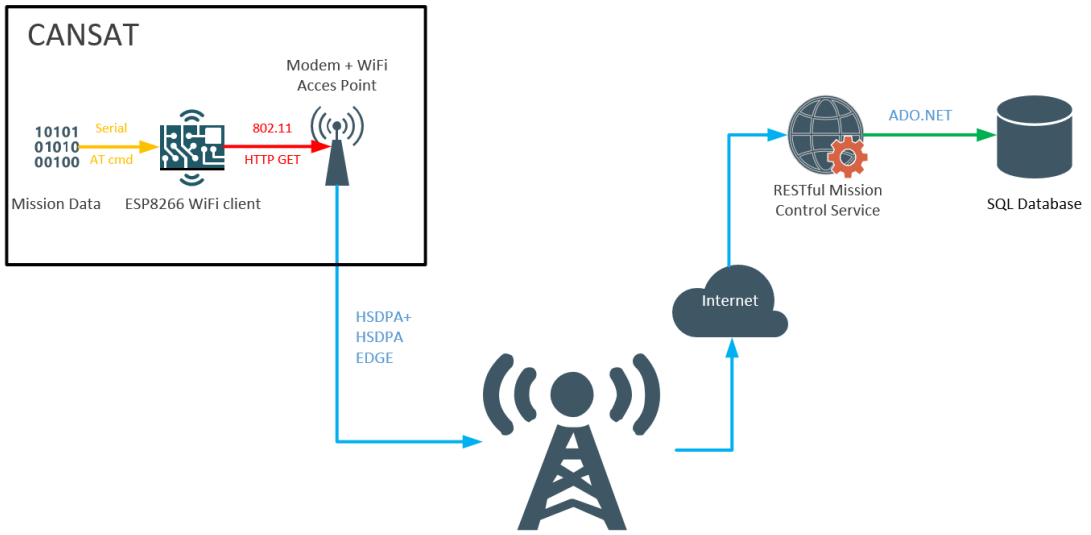


Figure 1.15 Schematic showing the functions of the retrieval system

The main goal of connecting Arduino to the Internet is to send the telemetry to a RESTful service [17] installed on a known Web Server. Then, the service will save that data in a relational database. We will use the following components for that goal.

- 1x ESP8266 Wi-Fi Module.
- 1x HSDPA modem + Wi-Fi access point.
- 1x Ld1117v33 linear voltage regulator.
- 1x 100nF capacitor.
- 1x 10uF capacitor.

We also need a Web Server with a public IP address, with port forwarding if necessary.

1.4.1 Design process

The Arduino Uno used for processing the mission data onboard the probe is not capable of connecting to the Internet by his own means. Therefore, external components need to be added to the device. First of all, our investigations based on the paper *UMTS-HSDPA in High Altitude Platforms (HAPs) Communications* [18] showed us that HSDPA networks (that provide 3G internet access to cell phones) have good availability in high altitudes, even above 20km. This availability is increased in rural areas, like the one that will be used for the FourCast launch. This is why we decided to use the HSDPA network in order to connect the FourCast microcontroller to the Internet.

As the availability of IoT Wi-Fi clients compatible with Arduino is much higher than HSDPA IoT clients, we decided to create a small Wi-Fi network inside the probe, using a HSDPA modem that also acts as a Wi-Fi access point, known commercially as MiFi modems [19] (see Figure 1.16). Choosing the HSDPA

modem is not complex, because we only had to focus on weight, size and battery life and because, actually, all models are quite similar.

Figure 1.16 Vodafone R216 HSDPA modem MiFi

The selection of the ESP8266 [20] (see Figure 1.17) (view Annex A) module was done under the following characteristics:

- One of the cheapest IoT modules in the market.
- High stability over large variety of 802.11 networks.
- Automatic reconnection.
- Rich and customizable firmware, with lots of protocols default enabled (DHCP, DNS, TCP, UDP…).
- Serial interface, ability to control it using large variety of AT commands.
- Ease of interaction with any kind of microcontroller.

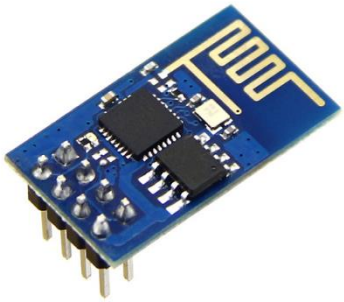


Figure 1.17 ESP8266 Wi-FI module

The ESP8266 module power ratings state that it can consume up to 150mA at boot time. This is much higher than the current which Arduino can provide (max 50mA). This could be the cause of undesired behaviors, like random disconnections or unexpected shutdowns. This is why the ESP8266 needs to be feed directly from the battery, using a 3.3V regulator [21] (view Annex A).

The capacitors are used to avoid overloading and for increasing the efficiency of the linear voltage converter. They must be connected as shown in Figure 1.18.

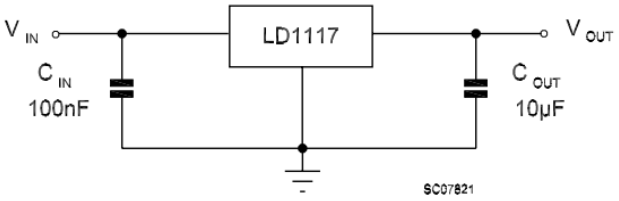


Figure 1.18 3.3 Voltage regulation

We can connect the Tx and Rx serial ports of the ESP8266 to any of the customizable digital ports of Arduino, but we need to take careful record of this information in order to configure the *SoftwareSerial* library afterwards. In some ESP8266 boards the CH_PD pin needs to be connected to 3.3V also. The Tx pin is labeled UTXD and the Rx pin is labeled as URXD (see Figure 1.19).

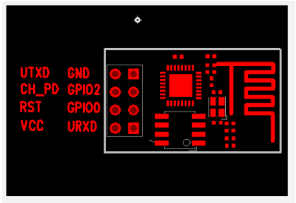


Figure 1.19 ESP8266 pin function

The Arduino code used to send the HTTP petitions to the server is shown in Annex B.

1.4.2 WCF RESTful service

In order to receive the petitions made by the ESP8266 in the FourCast, we need to create server code. The main purpose of this code is to create the interface needed to communicate with the FourCast, and to insert the data that it sends into a SQL database. In order to do that, a computer equipped with Windows will be accommodated to host the RESTful service, using Internet Information Service (IIS) [22]. The reasons for choosing a Windows machine are:

- Fully integrated Visual Studio 2015 solutions, that can be deployed directly into IIS.
- Existing templates for RESTful interfaces in Visual Studio 2015.
- Free licenses for Visual Studio 2015 using the UPC student account.
- The developed code can be installed in any Windows computer (easy available)

The Windows machine where the service will be implemented will need to meet the following requirements:

- .NET Framework 3.5 or above:
- Port forwarding (preferably 80 TCP)
- Non-blocking Firewall.
- SQL Server (Express versions are free)

Besides above requirements a few Windows packets need to be installed, using the “Add Features” option inside “Programs” at Control Panel:

- .Net Framework 3.5
  - **HTTP Activation**
- Internet Information Services (standard selection)
  - **ASP**
  - **ASP.NET 3.5**

Once this features are correctly installed, the program IIS can be run and we can use the computer as a Webserver.

1.4.3 Server code

The full server code is included in Annex B, but in order to understand how it works and for didactic purposes the basics will be explained in this section.
The code is written in C#, and the project structure is shown in Figure 1.20.

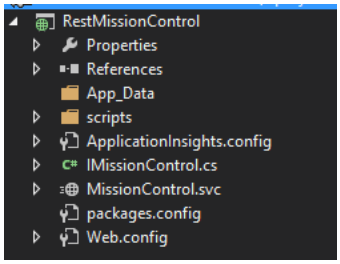


Figure 1.20 Code structure for the RESTFUL service

The most important files are:

- **MissionControl.svc**: Contains the code for the service operation that inserts the data sent by the FourCast into the SQL database.
- **IMissionControl.cs**: Interface that implements the operation included in the above file. It contains the definition of the operation, including inputs and outputs, and response format.
- **Web.config:** Contains information about the service endpoints and connection strings. It also contains information about the allowed methods (GET, POST, etc.) and the SQL database connection string.

We have defined only one service operation, which is the one that inserts the mission data into the SQL database.

1.4.4 SQL Database

In order to store the data, a SQL database will be used, with a single table and the following columns (see Figure 1.21).

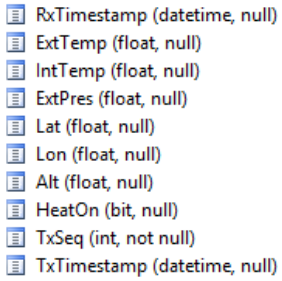


Figure 1.21 SQL Table columns

- RxTimestamp stores the timestamp when the packet is received at the server
- ExtTemp stores the external temperature (RTD sensor)
- IntTemp stores the internal temperature (LM35 sensor)
- ExtPres stores the external pressure (MPX2200AP sensor)
- Lat stores latitude (GPS)
- Lon stores longitude (GPS)
- Alt stores altitude (GPS)
- HeatOn: informs if the heating system is on (MOSFET switch)
- TxSeq: numerical sequence.
- TxTimestamp: timestamp when the packet is sent by the probe (GPS)

1.5 FourCast positioning system

The knowledge of the real-time position of the probe is an important asset that is needed for the success of the overall mission. Without it, the other measures provided would lack of consistency. It is known that temperature and pressure magnitudes vary as a function of height, therefore a system capable of reading height must be provided to give consistency and coherence to the measured magnitudes. Typically, altitude has been measured using barometric readings (see Figure 1.22). Altitude can be calculated knowing pressure's rate of change as a function of height and having also the sea level pressure as a reference. In the experiment, but, the pressure sensor should not be used to compute the altitude because we would enter in a circular process where pressure and altitude would be co-dependent. In order to obtain consistent data, we must use an independent source of information for altitude readings.

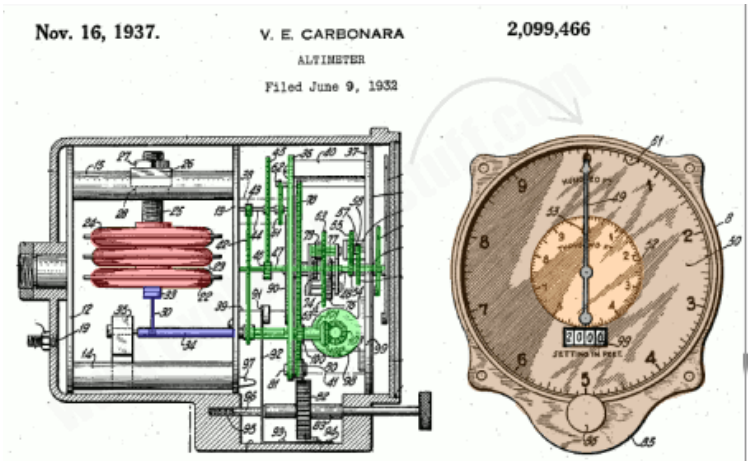


Figure 1.22 Pressure altimeter US Patent #2.099.466

Other kind of altimeters are RADAR altimeters, which can calculate altitude by using radio waves directed to the ground, measuring the latency between the emission and the comeback of the wave. This kind of systems require advanced antennas and therefore would not be suitable for the CanSat.

Luckily, nowadays it exists Global Navigation Satellites Systems (GNSS) that can provide accurate navigational information, three-dimensional position coordinates and a precise time signal. If we obtain this information continuously derived magnitudes such as speed or acceleration can also be computed, both in module and as vector quantities. If we provide the probe with a GNSS module it will be able to read three-dimensional accurate position data and a rich of other data:

- Altitude can be used to compare temperature and pressure readings to the ISA standards.
- Lateral position can be used to place in real time where the probe is plus estimate possible landing locations for recovery purposes.
- Vertical speed data can be used to estimate how long will the probe take to reach the desired altitude.

The GNSS system that will provide this information is the GPS. We have chosen GPS because it is globally available, it is very well documented and due to its easiness for finding resources and modules.

1.5.1 Fundamentals of Global Positioning System (GPS)

Developed during the mid-70s by the U.S Military, and opened to civilian use after the incident of Korean Air Lines Flight 007, that was shot down in soviet air space [23], GPS provides a global constellation of satellites that, along with triangulation principles can be used to compute users' positions. Using information included in navigation messages broadcasted from at least 4 GPS satellites constellation, a receiver can estimate its position. A navigation message includes precise information about the coordinates of the GPS satellite that transmits it, and a

precise timestamp that marks when the signal was generated, referenced in GPS time, a common and precise clock reference along all the GPS network. As mentioned above, a receiver requires at least 4 satellites in order to compute its position, three in order to make the triangulation of the position and one to synchronize the time framework of the receiver with the common GPS clock onboard the satellites. The equations needed to be solved are shown in Figure 1.23.

$$P^1 = ((x^1 - x)^2 + (y^1 - y)^2 + (z^1 - z)^2)^{1/2} + c\tau - c\tau^1$$

$$P^2 = ((x^2 - x)^2 + (y^2 - y)^2 + (z^2 - z)^2)^{1/2} + c\tau - c\tau^2$$

$$P^3 = ((x^3 - x)^2 + (y^3 - y)^2 + (z^3 - z)^2)^{1/2} + c\tau - c\tau^3$$

$$P^4 = ((x^4 - x)^2 + (y^4 - y)^2 + (z^4 - z)^2)^{1/2} + c\tau - c\tau^4$$

Figure 1.23 Pseudo-range navigational equations.

More than 4 satellites can be used for estimating position, and in that case the pseudo inverse matrix has to be used in order to obtain a square matrix that can be used to solve the system. In this way, local errors are minimized, and a more precise estimation is provided.

The GPS navigational system is broadcasted to Earth in several frequency bands [24]. Most of the civilian receivers use the C/A signal, transmitted in the L1 band, at 1575.42MHz. The GPS receivers are built in with the known as COCOM limits [25]. Basically, the receiver stops working if it calculates that its position is above 18km or its speed is superior to 1800km/h. This limit was imposed in order to avoid using civilian GPS for guiding ICBMs or military purposes. The altitude limit is a major drawback for our experiment, since we will have to deal with dead reckoning navigation.

1.5.2 Obtaining position from GPS

For this project, we have decided to use the GY-NEO6MV2 GPS Module [26] (see Figure 1.24) (view Annex A).

Figure 1.24 The GY-NEO6MV2 GPS module.

This GPS module includes a firmware that processes the GPS navigation signal and computes the position equations. The resulting information from the computations is encoded into NMEA 0183 protocol [27] and transmitted via serial communication. This is an important change respect the pressure and temperature systems. The Arduino microcontroller will not do any computations this time, all the computations are done by the firmware of the receiver. The only thing that the Arduino must do now is to gather the information communicated via serial port from the GPS receiver. This is useful because it saves memory space and computational workload for the microcontroller.

We can feed the GPS module directly from the 3.3V or the 5V sources from Arduino. The GPS module works at 3.3V, but has a built-in voltage regulator, so it can be supplied with 5V also. The GND must be referenced with the Arduino GND and the Tx and Rx pins can be connected to any of the free use digital ports of the Arduino (2-7).

Therefore, we will need a serial bus emulator, in order to provide Arduino an interface to communicate with the GPS module. The library *SoftwareSerial* [28] accomplishes that. *SoftwareSerial* can convert two any digital ports into a serial bus. It has its limitations, for example, two serial buses created with this library can't communicate simultaneously. Figure 1.25 shows the connections that we will do.

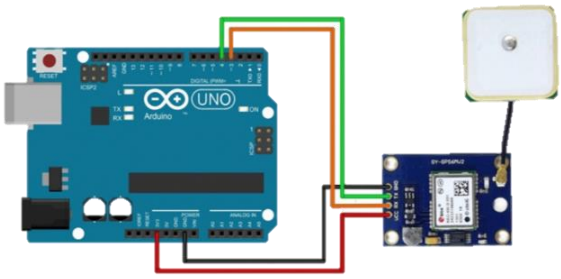


Figure 1.25 GPS module connections with Arduino

The code implemented in the microcontroller can be found in Annex B.

We can see that the GPS module communicates the position information via NMEA 0183 sentences. The serial monitor will look like Figure 1.26.

```
$GPGSA,A,1,,,,,,,,,,,,,,*1E Datatype = GPGSA COUNT(Terms) = 19
$GPGSV,3,1,12,05,09,115,00,06,12,221,34,09,11,045,00,15,70,053,00*7F Datatype = GPGSV COUNT(Terms) = 21
$GPGSV,3,2,12,16,05,241,33,18,51,266,35,21,68,209,00,22,11,284,35*79 Datatype = GPGSV COUNT(Terms) = 21
$GPGSV,3,3,12,26,35,131,00,27,15,053,00,29,40,356,34,30,00,267,00*74 Datatype = GPGSV COUNT(Terms) = 21
$GPGLL,,,,,,V,N*64 Datatype = GPGLL COUNT(Terms) = 9
$GPBOD,,T,,M,,*47 Datatype = GPBOD COUNT(Terms) = 8
$GPVTG,,T,,M,,N,,K*4E Datatype = GPVTG COUNT(Terms) = 10
$PGRME,,M,,M,,M*00 Datatype = PGRME COUNT(Terms) = 8
$PGRMZ,,f,1*29 Datatype = PGRMZ COUNT(Terms) = 5
$PGRMM,WGS 84*06 Datatype = PGRMM COUNT(Terms) = 3
$GPRMC,,V,,,,,,,180712,19.3,E,N*0E Datatype = GPRMC COUNT(Terms) = 14
$GPRMB,V,,,,,,,,,,,,A,N*13 Datatype = GPRMB COUNT(Terms) = 16
$GPGGA,,,,,,0,00,,,M,,M,,*66 Datatype = GPGGA COUNT(Terms) = 16
$GPGSA,A,1,,,,,,,,,,,,,,*1E Datatype = GPGSA COUNT(Terms) = 19
```

Figure 1.26 Examples of NMEA 0183 sentences

As we can see, position information is encoded into $GPRMC sentences. In order to parse all the information from the NMEA 0183 sentences into concrete variables (latitude, longitude and altitude) we will use the *TinyGPS* [29] library. *TinyGPS* is just a NMEA parser, it reads NMEA sentences and obtains the information from them, without the tags and other particularities of the protocol. It also includes some more advanced functions, like distance calculation between two points, speed indicator and more.

1.6 FourCast heating system

The heating system was not expected to be in our FourCast at the very beginning; however, we realized it was necessary to include it in order to control the internal temperature and avoid negative values which might affect the electronics. There are two subsystems inside the heating system. On the first hand, we have a sensor that controls the internal temperature. On the other hand, we have a sort of heater necessary to protect the electronics. Both systems require few electronic components, which is convenient to us, as they will not take up much space inside the FourCast.

The components that will be used to design these systems are listed below (view Annex A for datasheets):

- A temperature sensor (LM35) [30]
- 2x 100Ω resistors of 1W
- A MOSFET transistor [31]
- An aluminum plate

The first system is very easy to design as it only consists of the temperature sensor. This sensor will be fed by the microcontroller and will show as an output value the voltage that corresponds to the surrounding temperature. The sensitivity of this sensor is 10mV/ºC so, the corresponding voltage value must increase 100 times in order to get the resulting internal temperature.

The second subsystem consists of the solution we propose in order to avoid freezing the electronics (most components do not operate at temperatures near or below 0 ºC). To avoid this condition, we decided to build a heater (see Figure 1.27). We can appreciate in this Figure that a cable is not connected. This cable should be connected to the microcontroller's PWC pin.

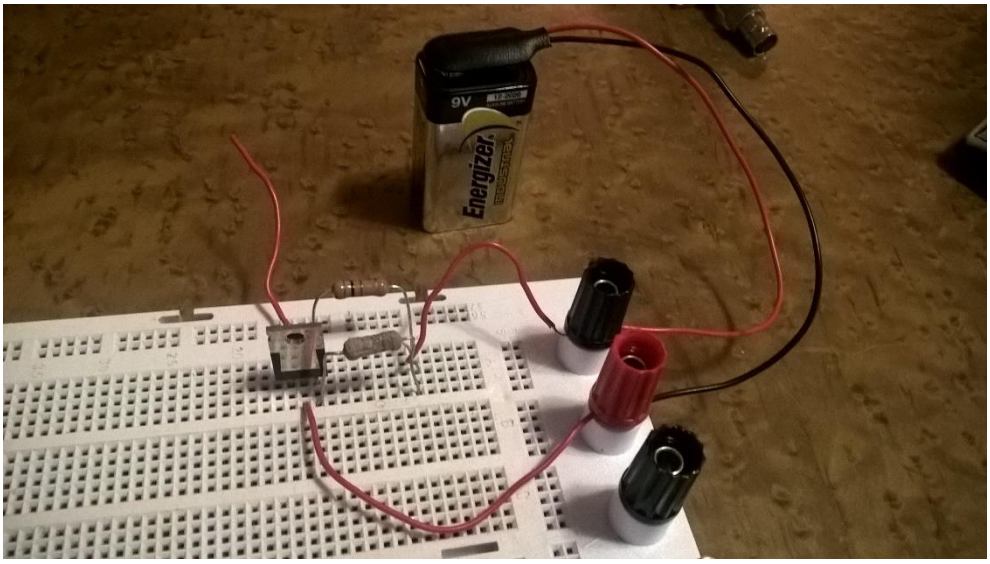


Figure 1.27 The heating system

As we do not want to heat all the electronics during the whole time of the mission, we decided to implement a MOSFET that works as a switch. This MOSFET should be used went the temperature is below a given value. The electronics can be heated by letting pass current to the 2 resistors connected in parallel. To dissipate enough heat, and in a higher area, we decided to place an aluminum plate (see Figure 1.28) on top of both resistors. This second subsystem takes as reference lectures, the values extracted from the internal temperature sensor. If the value read by the temperature sensor is higher than the one chosen as the switch value, the transistor will stop working and the resistors will stop heating. We have set 2ºC as the temperature below which the heater is to be turned on, whereas it is not until the temperature is above 3ºC that the MOSFET will be turned off.

Figure 1.28 Aluminum plate.

It is interesting to calculate the current used by these two resistors in order to know the power required. Considering that the voltage received by both resistors is 9V, we will have the following calculations:

$$R_{eq} = \frac{R_1 . R_2}{R_1 + R_2} = 50\Omega \quad \textbf{(1.24)}$$

$$I = \frac{V}{R_{eq}} = 180\ mA \quad \textbf{(1.25)}$$

$$P = V . I = 1{,}62W \quad \textbf{(1.26)}$$

We will have 1,62W for both resistors so 810mW per resistor.

1.7 FourCast power supply system

The power supply system is crucial for our probe, because it provides the power to boost the entire system. Depending on the characteristics of the battery more components might be added and the probe's autonomy would be modified. Besides, the choice of a battery may affect the final mass of the system. It is very important that the chosen battery gives enough power to all the systems described in the previous sections, as well as enough autonomy for the whole mission.

Building the power supply system is not difficult: we have to choose a suitable battery and then connect it to the microcontroller. We have different options in terms of battery choice. We can choose between rechargeable or non-rechargeable batteries, and between batteries with different voltages. Our first option was a non-rechargeable battery, because. this type of batteries are the most common ones, and because they can be lighter than the rechargeable ones. In addition, we had to select the battery depending on its voltage and capacity.

The first thing we needed to do was to verify which is the voltage that can be supplied to our microcontroller. In our case, we could have a supply voltage between 7 and 12V. If we wanted to feed our microcontroller with a single battery, we could use a 9V battery. However, the main drawback of these batteries is their low capacity with respect to smaller non-rechargeable ones. As an alternative, we could use AA alkaline batteries, but their low voltage (1.5V each) meant we would have needed at least 5 of them, which meant an unnecessary increase in mass and volume quite against the philosophy of this kind of devices. This meant that the 9 V battery was the best option in our case to supply the power necessary for the entire probe.

The components of the power supply system are a 9V Energizer battery [32] (view Annex A), which has a capacity of 625mAh and a battery retainer clip, which in our case is a Bud Industries battery retainer (see Figure 1.29) [33].

Figure 1.29 9V battery with retainer clip

In order to calculate the nominal capacity required by FourCast we need to know the operating current of its different components. As a summary, these components are: the Arduino UNO, the pressure sensor (MXP2200), the internal temperature sensor (LM35), the external temperature sensor (RTD1000), the GPS, the XBee PRO, the voltage doubler (MAX660) and the recovery system (ESP8266).

It must be noted that there are some elements that do not require constant current throughout time; this is the case of the GPS, the recovery system or the XBee PRO. The reason for these different current values is that they have different working modes: transmission, reception, sleep mode, etc. The XBee PRO will always be transmitting during the mission, so it will always operate at maximum current, however, the recovery system will only transmit 3s per minute and the remaining 57s will be in sleep mode; in this case, we calculate the mean current. GPS will be working continuously, so maximum current will be used for the calculations.

The following table shows the current taken from each element from their corresponding datasheets (Table 1.3).

Elements	Current used (mA)
Arduino UNO [34]	46
MXP2200	6
LM35	0.091
Pt1000	0.45
GPS	67
XBee PRO	65
MAX660	0.120
ESP8266	21.25
Voltage doubler	10

Table 1.3 Current used by elements in the probe.

Once we have these values we must add them up to get the total required current of the system (215.9mA). We can get the autonomy of the system:

$$Autonomy = \frac{Battery\ Capacity}{Total\ CanSat\ capacity} = \frac{625mAh}{215.9mA} = \mathbf{2.91\ h} \qquad \mathbf{(1.27)}$$

The current required by the heating system is high, but difficult to determine, as we cannot a priori tell for how long it will be operating (we merely know that it will be on when the room temperature goes below 2ºC). This is the reason why it has not been included in the equation 1.27, as it is not certain. Nonetheless, we can estimate operation time of the system with the data obtained from tests described in chapter 3 of this document. The temperature inside the probe will remain above 2ºC around 2.5 hours since launch. At that point and at the rest of the flight the heating system will switch on and off, approximately being active 50% of the time. If we take the calculations in equation 1.25 and estimate that the system is active 25% of the FourCast flight and guess that total flight time is 6 hours, the capacity drained by the system is approximately 67.5mAh. We estimate this could shorten battery life between 10% to 15% respect the previous calculations.

In order to make sure that the probe will have enough autonomy we have traded charge by mass and included an additional PP3 battery to be connected in parallel to the first one. By connecting the batteries through a blister, we get two different outputs: one to feed the microcontroller and another to will feed the recovery system.

1.8 FourCast Arduino UNO microcontroller

All the data coming from the different sensors has to be acquired and converted into a digital quantifiable signal, in order to obtain real measures on the read magnitudes. Also, it is necessary to communicate with the digital devices, such as the GPS or the ESP8266. There are multiple programmable types of devices that can achieve all these things, each one with singularities that make him appropriate or not for the experiment. Mainly, we could choose between two alternatives:

- Microcontroller (such as Arduino, Funduino, etc)
    - Low power consumption (50mA)
    - Low computational power
    - Small and compact.

- Microprocessor (such as Raspberry Pi)
    - Bigger computational power
    - Huge power consumption (1.5A)
    - May need peripherals.

In our case, the choice is quite clear, as we do not need to make elaborated calculations and power consumption is one of our main limitations. We only need to gather analog data from the sensors and handle communications with two devices, using serial interface. This is why the most appropriate choice is a microcontroller. We chose Arduino UNO [35] (see Figure 1.30) (view Annex A), which features an ATmega328P microcontroller with 14 digital I/O and 6 analog inputs. It has a clock speed of 16MHz and 32kB of EEPROM for storing the instructions. As the code that we need to write for our purpose is quite simple, we do not need higher memory and therefore Arduino UNO makes the perfect choice. Also, it is a very common model and there are lots of accessories that can fit into it, like the XBee shield. This also makes the whole project cheaper.

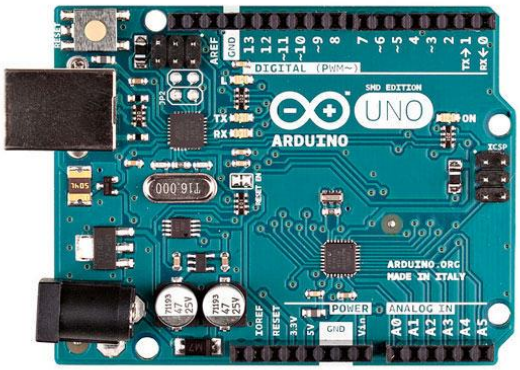


Figure 1.30 Arduino UNO board like the one used at the project

1.8.1 Digital low pass filter

During the design phase, we put a lot of effort in reducing the noise on the analog sensor systems, designing first order and second order low pass filters. Second order filters require more components and, if they are designed as active filters (such as second order Sallen-Key) they also have a power consumption component that cannot be neglected. Doing multiple tests, we found out that most of the noise was added during the quantification process on the Arduino UNO microcontroller. This is because the sampling is done at 1.1V (it is a low voltage, and a few mV of noise can alter the result significantly) and because the quality of the DAC is not very good (there exists better devices). Taking this into account, we realized that the most effective way to reduce noise was performing multi sampling in the Arduino code, working as a digital low pass filter. In our case, it resulted more convenient to take 15 samples of pressure during 3 seconds and take the mean value than to add higher order low pass filters. This is why we only use first order filters. We can afford to use these first order filters because we do not need to send data at very high speed, as that kind of low pass digital filter consumes processing time.

1.8.2 Considerations about the digital I/O

As explained in section 1.5 (FourCast Retrieval and enhanced communications system), the ESP8266 is feed directly from the batteries, and not from the Arduino, due to current requirements. Digital communication in Arduino UNO microcontroller is at 3.3V, so it is very important that the battery GND pole and the Arduino UNO GND are shared. If both GND are not put in common, then the digital I/O pins do not read correctly the data sent through them. In order to achieve that, it is necessary to physically join the two GND, using cables.

1.9 Parachute

There are several ways to lift the probe into the upper layers of the atmosphere. We decided to use a helium balloon to do so, mainly because we do not need a high climb rate and it's simpler than using a rocket. In chapter 4 we will describe the particular helium balloon used in our case for the experiment.

As the balloon holding the probe increases its height the atmospheric pressure decreases and, eventually, becomes much lower than the gas pressure inside the balloon. At this moment, the balloon explodes and the FourCast is expected to start a free-fall back to the ground. In order to slow-down the falling velocity and avoid risks we decide to include a parachute.

After some research, we have seen that there are many types of parachutes, for instance cross shaped, flat and semi-spherical. We finally decided to implement the flat parachute [36]. Our first idea was to buy the parachute at a modeling shop, however, it was not available in Barcelona with the characteristics we needed. Further research in the existing literature lead us to opt for designing our parachute with an ordinary umbrella.

Two forces are the ones acting on a parachute when falling. On the one hand, we have the gravity force, which will decelerate the system, and, on the other hand, we have the drag force from the parachute that will therefore reach equilibrium with the gravity force. With this equilibrium, acceleration will become zero and the device will descent at a constant velocity.

We also had to decide the dimensions of the parachute, and so using concepts from basic mechanics:

$$\textbf{Gravity force: } F_g = m * g \qquad \textbf{(1.28)}$$

$$\textbf{Drag force: } F_D = \frac{1}{2} * \rho * V^2 * C_D * A \qquad \textbf{(1.29)}$$

In these equations, we have:

M: mass of the probe.
g: acceleration of gravity, equal to 9.81m/s^2
ρ: air density, consider it as the sea level density equal to 1.225kg/m^3
V: descent velocity of the probe.
C_D: drag coefficient of the parachute.
A: total area of the parachute.

Our parachute (an umbrella) can be approximated by a heptagon inscribed in a circumference of radius 0.53 m and thus its corresponding area is simply 0.769 m^2 (see Figure 1.31).

Figure 1.31 Parachute with its ropes

The velocity of an object falling under the effect of gravity and friction expressed as above is:

$$a = \frac{dv}{dt} = -g + \frac{b}{m}v^2, \qquad \textbf{(1.30)}$$

with

$$b = \frac{1}{2}\rho C_D A \qquad \textbf{(1.31)}$$

This yields the expression $v(t) = \sqrt{\frac{mg}{b}}\tanh(\sqrt{\frac{bg}{m}}\ t)$

$\sqrt{\frac{mg}{b}}$ is the limit velocity, v_{lim}, that is, the velocity of the object at time tending to infinity, and equals to the velocity achieved when gravity and drag balance each other. This value is technically reached at time tending to infinity, but at a time as short as t=1.4s the velocity of the object is already 0.999 v_{lim}. Therefore, it is a good approximation to assume that the entire fall occurs at v_{lim}, and thus:

$$m * g = \tfrac{1}{2} * \rho * V^2 * C_D * A \qquad \textbf{(1.32)}$$

$$V = \sqrt{\frac{2 * 0.7\ Kg * 9.81\frac{m}{s^2}}{1.225\frac{Kg}{m^3} * 1.25 * 0{,}7687\ m^2}} = \mathbf{3.41\frac{m}{s} = 12.3\frac{km}{h}} \qquad \textbf{(1.33)}$$

This velocity is very acceptable. It is not big enough to cause any damage and it is not small enough to make the FourCast glide away and move a long distance

from the launch location. Considering that the expected maximum height of the FourCast is about 35km and that it will descend at constant velocity we can easily infer that it will take about 4 hours to land after the deployment.

Deploying the parachute was another delicate issue. First, we had to decide how to place and attach the parachute and the FourCast. It could either be free of retained somewhere so that the deployment will be mechanical or electronic. We have finally decided to place it freely (no mechanical or electronic deployment) but attached to the balloon. It will be placed between the FourCast and the balloon so that when the balloon explodes our parachute it will deploy automatically.

CHAPTER 2. FOURCAST ASSEMBLY

2.1 Assembly process

In this chapter, we will talk about how we managed to fix and assemble all our components into the probe structure. We will focus on the different materials we have used to do this assembly, going from Dupont cables to soldering. Furthermore, we will see the design we have done for the system and how we have introduced all the different systems inside it. We will also explain one significant feature of our device, which is its modular design.

The number of systems that make the probe and the fact that components used are quite big in size, for example the XBee shield, make it impossible to fit it inside regular CanSat dimensions. As this project is not a part of any competition, just for scientific interest, we took the decision to make it slightly bigger and heavier. This decision gives us more freedom of design and also allow us to use all the systems previously described.

2.1.1 Materials used

Our first approach will pay attention on the different cables used, the soldering and the electronic boards. After designing all our electronic systems into a Protoboard to do the first tests, we started to think of a way of packing all these elements into the probe. Our first thought was using printed circuit boards with all connections already build-in. We would only have to weld the components but the most difficult part would already be executed. Nonetheless, by the time we ended all our designs it was August and we could not print those boards because the campus technical services were closed. That is why we decided to take simple boards and weld by ourselves all the circuits.

When we designed our systems in the Protoboard, we saw that they took a lot of space because we had many cables going from one place to another and this compromised space. We wanted to reduce space to its minimum in order to reduce weight and the dimensions of the probe. The positive thing about the simple electronic boards is that the entire connections can be welded and there is no need for cables anymore, this was a huge advantage because we saw the occupied space decrease a lot. Nonetheless, the soldering of the components and subsequently the connections were not easy tasks considering that we had we had never welded before. It took us a lot of time to sold all the systems, because tin is not easy to manage the welder and the tin and it can sometimes go to undesirable places.

As we said, we wanted all the systems to be very compact in order to reduce used volume; this increased the difficulty of the soldering. We first started with the external temperature system (red) because it was the one that had more components and the higher number of connections (see Figures 2.1 & 2.2). Continuing with our goal of reducing space, we decided to add two systems together with the external temperature system in one same board: the ones chosen were the pressure system (blue) and the internal temperature system (purple).

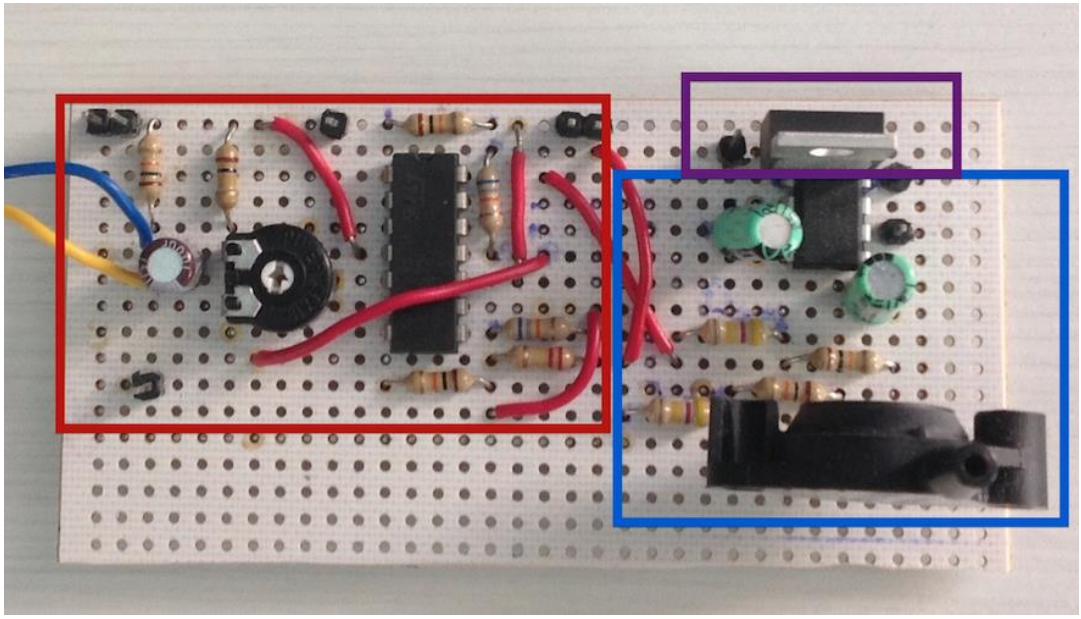


Figure 2.1 Board 1 front's view.

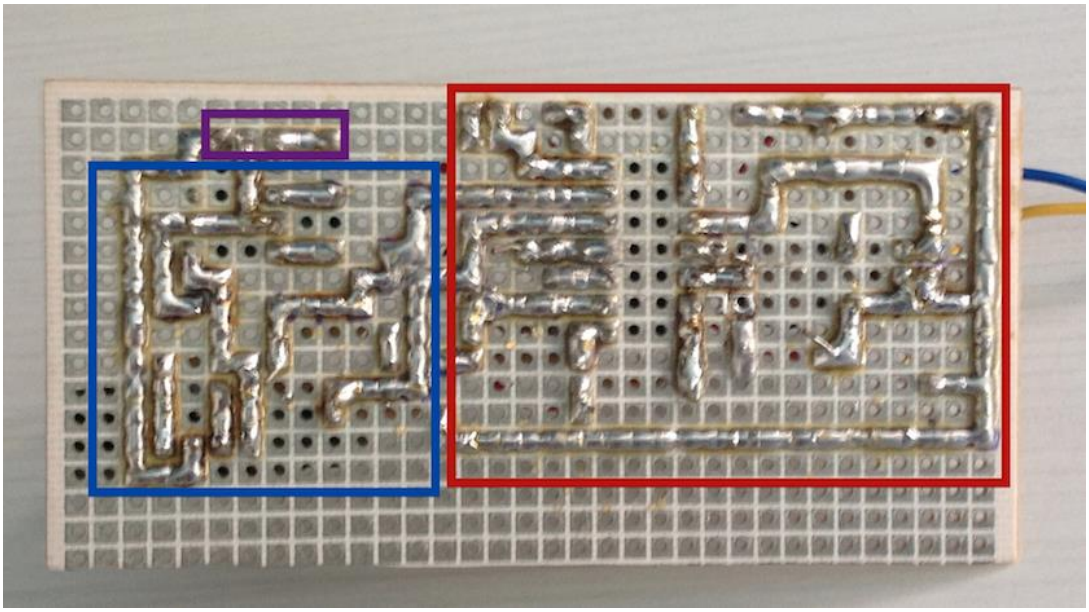


Figure 2.2 Board 2 rear view.

Once we had this first board welded and in order to not to saturate all systems, we decided to take a second board. This second board carried the heating system and the retrieval system. Moreover, we also thought that this would be a good idea because we could place both boards face to face so that when the conditioning system worked it would heat up all the electronics in Board 1. Board 2 (see Figures 2.3 & 2.4) was smaller than the first one because both systems had fewer components, which made it also easier to weld them.

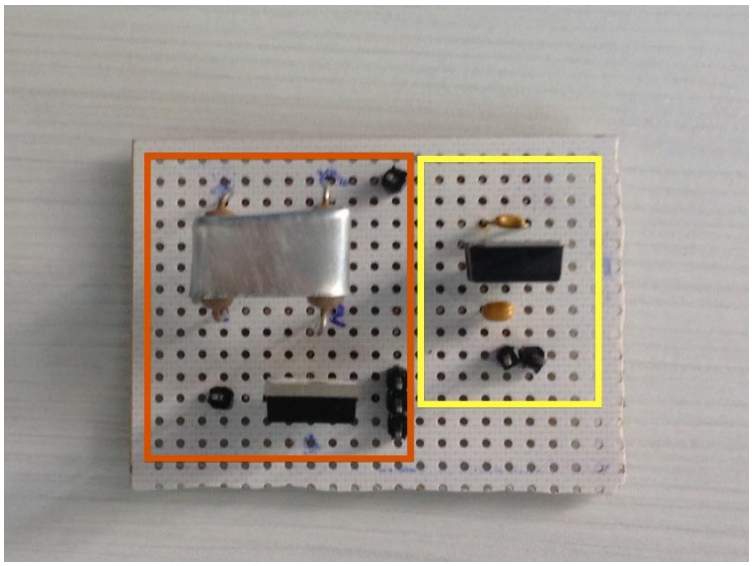


Figure 2.3 Board 2 front's view.

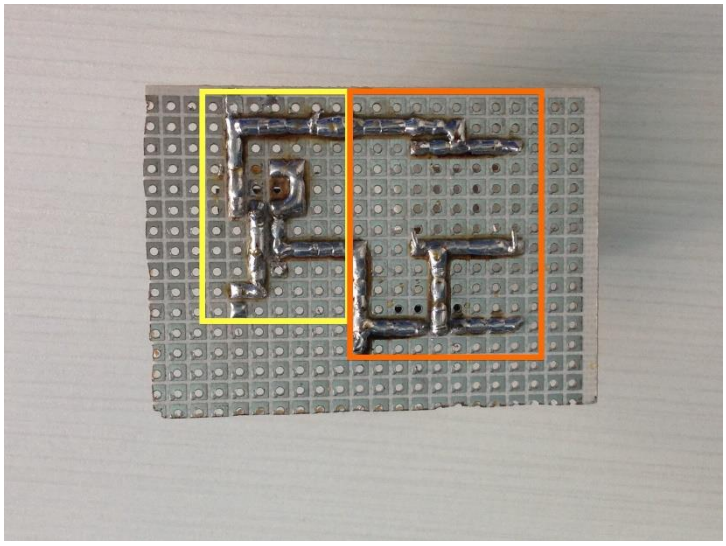


Figure 2.4 Board 2 rear's view.

While welding these two boards we had some problems. The first problem reported was that we could regulate the welder temperature in order to speed up the soldering. However, if it was too warm it was not beneficial for us because we could not control the tin. On the other hand, if the welder was not that warm we could weld any component. We had to find an intermediate temperature where we could weld without burning the board and we could have a more accurate control of the tin too. When we finished welding both boards we measured if the system was working, however, sometimes we had incorrect values in our tester. The difficult part in this process was to check if all tin connections were done correctly and if they did not touch with other components. It was not easy because sometimes it was only a lack of tin in a specific place or that the tin touched other connections in the board so we had to use the welder again to manage the tin.

We also had to remove some tin sometimes because the connection was not well welded.

Furthermore, after having checked that all connections where correct, we thought about the connections that had to be made between the microcontroller to the boards. We could had used the same cables we had when we designed at the very beginning all the systems but we were not convinced of their functionality and flexibility. Therefore, we tried to find new cables in order to do better connections. While looking for information into other projects [37] we find out that there was a type of cables very useful for this kind of missions: the Dupont cables (see Figure 36). These cables are very flexible and make very good connections. In each board, we have single connectors in order to bring these cables from the microcontroller to the boards. Each connector has a different functionality, such as power supply, ground or getting output values among others.

Figure 2.5 Dupont cables

2.2 FourCast assembly procedures

Once all components are correctly placed into their respective PCB and the connections have been tested, it is time to accommodate it into a common structure. It is important to keep all components closed up together, as it reduces the required space and it also helps controlling the temperature inside the probe. The design of the probe structure has been made according to the following principles:

- Cylindrical shape (in order to seam a soda can)
- Easy to assembly – disassembly (removable parts)
- Modulable (easy to add or remove components and systems)

- Keep important systems closed up together (for example, the heater close to the PCB board with the sensors and the operational amplifier).

Keeping these principles in mind, the design process can be started.

2.2.1 Components to accommodate

The whole probe systems are divided into 7 separate parts:

- **PCB 01:** PCB circuit formed by the RTD temperature sensor, the LM35 temperature sensor, the MPX2200AP pressure sensor and all the required electronics for the accommodation (capacitators, an operational amplifier and a voltage inverter).
    - **Inputs:** 5V and GND, all directly from Arduino. (2 pins).
    - **Outputs:** External temperature, external pressure and internal temperature read by Arduino analog inputs (3 pins).

- **PCB02:** PCB circuit formed by the LD111v33 voltage regulator and the heating system (resistors with aluminum plate)
    - **Inputs:** 9V from battery, GND (from battery and Arduino), PWM control from Arduino (4 pins)
    - **Outputs:** 3.3V regulated voltage (2 pins)

- **GPS:** GPS module with the ceramic antenna
    - **Inputs:** 3.3V and GND from Arduino (2 pins)
    - **Outputs:** Rx and Tx serial communication (2 pins)

- **ESP8266:** ESP8266 Wi-Fi Client
    - **Inputs:** 3.3V regulated from the LD111v33, to the CH_PD and V_{cc} pin and GND (3 pins)
    - **Outputs:** Rx and Tx serial communication (2 pins)

- **Arduino Uno with XBee shield and antenna:**
    - **Inputs:** 9V from the batteries and all the outputs above
    - **Outputs:** all the inputs above.

- **2x 9V batteries:**
    - **Outputs: 2x 9v/GND connectors:** (one supplying the Arduino and the other one for the PCB02 board inputs)

- **Vodafone R210 MiFi Modem**
    - No inputs or outputs

All these parts can be seen separately in Figure 2.6.

Figure 2.6 All FourCast Components

2.2.2 Design process

As we need a specific chassis to suit all the requirements, we decided to create our own 3D model, instead of looking for already constructed structures in special shops. This gives us the following benefits:

- **Precision**: We can create slots very precisely, and also tailor screws' positions. Besides that, the size of the container can be exactly as we need, making it extremely compact and therefore lighter.
- **Materials**: As we can print it in a 3D printer, we can choose the type of plastic that satisfies our needs, taking into account several factors, such as thermal conductivity.
- **Cost**: As the school owns a 3D printer we can use it in order to save money without looking at external providers.
- **Flexibility**: We can save the 3D model and other people can modify it in the future to create another probes.

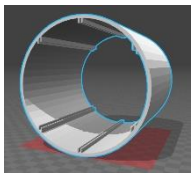
Then we decided to create a cylindrical shaped structure, formed by two cylinders, one inside the other. The gap between the external and the internal cylinder will be filled with thermal insulation material. We decided to create slots in the internal cylinder, where flat walls can be placed. In the walls, the different components can be placed using cable ties, making the required holes with a drill. Both sides of the walls can be used to place the components, in order to optimize the use of space.

This design concept raised three main questions, after answering them we could have the characteristics of our cylinder: diameter and height.

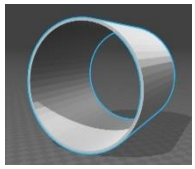
- How many walls do we need in order to accommodate the components?
- How much space there must be between the walls in order to allow room for the components and the wires?
- What should be the height of the walls?

There is not a single correct answer, as there exist multiple configurations that will satisfy our needs. We measured all the components described in the previous section and planned their position in the different walls. After many designs, we used the one that convinced us the most, formed by the following pieces:

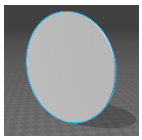
- **Internal cylinder**:
  - **Radius**: 6,25 cm.
  - **Height**: 14 cm.
  - **Thickness**: 3 mm.



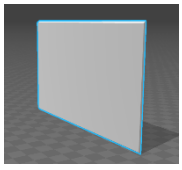
- **External cylinder**:
  - **Radius**: 6,75 cm.
  - **Height**: 14 cm.
  - **Thickness**: 3 mm.



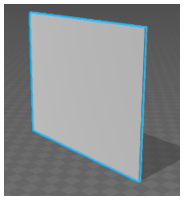
- **Cover (x2):**
  - **Radius:** 6,75 cm.
  - **Thickness:** 3 mm.



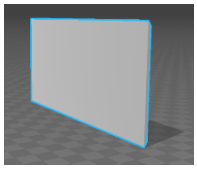
- **Wall A:**
  - **Wide:** 8,18 cm.
  - **Height:** 14 cm.
  - **Thickness:** 1,6 mm.



- **Wall B:**
  - **Wide:** 11,34 cm.
  - **Height:** 14 cm.
  - **Thickness:** 1,6 mm.



- **Wall C:**
  - **Wide:** 6,15 cm.
  - **Height:** 14 cm.
  - **Thickness:** 1,6 mm.



2.2.3 3D Printing materials

The plastic structure of our experiment was specifically designed for its contents, and printed using PLA plastic. We think it is important to remark the importance of a well-done design in a 3D CAD software such as Solidworks. It is important to know the characteristics of the printer that will do the job. In the design software, the piece always look perfect and has the lengths specified by the maker, but the reality is that the 3D printer has margins and tolerances, and therefore the finish of the piece is not perfect. It is very important to keep this in mind during the piece design, giving enough separation space between multiple solids, in order to allow slots or other structures to work properly.

The material chosen for the printing process is also important. It exists two type of plastics for generally available 3D printers: PLA and ABS. PLA is more common and it is also easy to work with. It melts at lower temperatures and its filaments are easy to manage, and it does not leave much dirt into the printer injectors. Despite that, this type of material is really porous and can present cracks at extremely low temperatures. As it is a porous material, its thermal conductivity is also a negative issue.

ABS was a more appropriate material, having a lower thermal conductivity (which is good for thermal control issues), and a better surface finishing. The mechanical properties of ABS are also somewhat better than for PLA (specially strength, flexibility and machinability).

Finally, we decided to use PLA instead of ABS because we did not have enough resources, and could not find a suitable vendor in the time we had.

2.2.4 Components distribution

The different components have to be placed into the walls, in a way that guarantees that they do not interfere with each other. Therefore, beside of the drills needed to accommodate the cable ties and fix the components, additional drills have to be made to pass the cables between the walls. The final distribution of the components is as follows:

2.2.4.1 Wall A

- **Side A1 (facing internal cylinder, see Figure 2.7):**
  - The two 9V batteries, placed vertically one above the other. One cable tie fixes each one horizontally crossing them.
  - Cable Hole 01: placed vertically aligned with the batteries. This hole allows the battery clip cables to pass to the other side.

- **Side A2 (facing side B1, see Figure 2.8)**:

- Arduino + Shield + XBee, fixed with 2 cable ties using the existing Arduino screw holes. It is placed near the bottom down, in order to allow the antenna to be located outside the probe.

2.2.4.2 Wall B

- **Side B1 (facing side A2, see Figure 2.9)**:
  - GPS, fixed with 2 cable ties using the existing GPS screw holes.
  - ESP8266, fixed with a single cable tie, crossing it horizontally.
  - Cable Hole 02: placed under the GPS, this hole enables a straight path to the Arduino analog inputs.
  - Cable Hole 03: placed above and between the GPS and ESP8266. This hole enables straight path for the battery cables that need to go to the side B2.
- **Side B2 (facing side C1, see Figure 2.10):**
  - PCB02, fixed with 2 cable ties using drilled holes were no components or tracks were. It is placed behind the GPS.
  - Vodafone R201 MiFi modem, fixed using 2 cable ties, placed in front of to the PCB01.

2.2.4.3 Wall C

- **Side C1 (facing side B2, see Figure 2.11):**
  - PCB01, fixed with 2 cable ties using drilled holes were no components or tracks were. It is placed in front of PCB02 and the MiFi modem, as it needs the heating system that contains it.
- **Side C2 (facing internal cylinder):**
  - Empty.

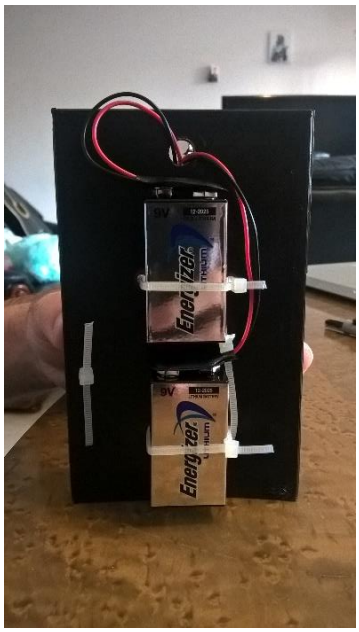


Figure 2.7 Side A1

Figure 2.8 Side A2

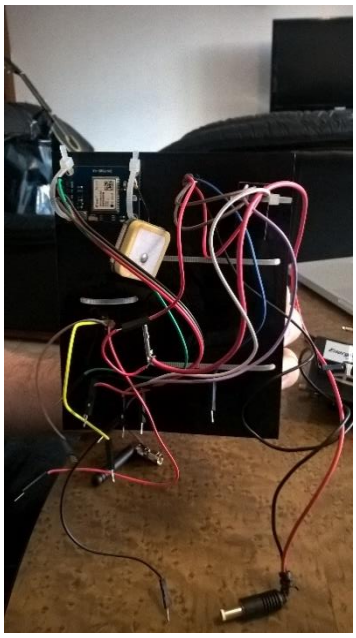


Figure 2.9 Side B1

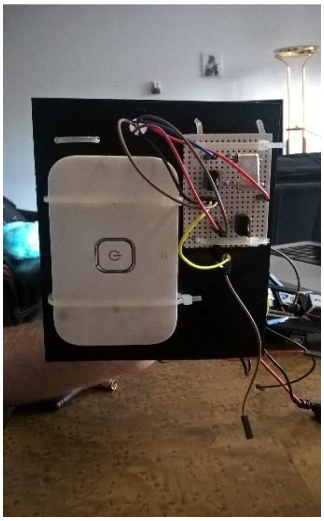


Figure 2.10 Side B2

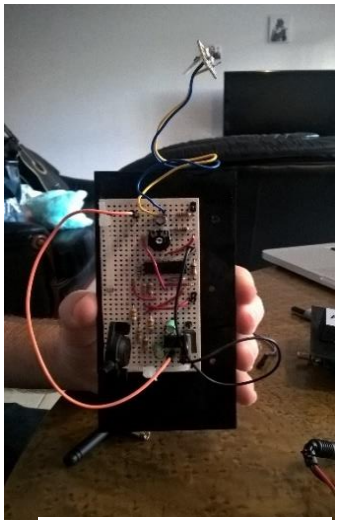


Figure 2.11 Side C1

2.2.5 Additional customizations

Two additional customizations have been made into the structure in order to complete the functionality:

- **Antenna Hole:** Placed in the bottom cover It had to be perfectly aligned with the XBee SMA connector. In order to place it correctly we attached some blue tack adhesive into the XBee SMA connector and make it hit the bottom cover. Then the blue tack adhesive stood in the bottom cover and we knew exactly where to drill.
- **Parachute ring:** Placed in the top cover, it had to be placed near the center of gravity, but also in a position where does not interfere with any

device inside the probe’s structure. Finally, the hole was placed near wall B, side B1, between the GPS and the ESP8266.

CHAPTER 3. TESTS AND VALIDATIONS

In this chapter, we will be talking about the different tests and validations we had performed in order to prove that the design we have created works the way it should. We have separated most of the systems but some of them are analyzed together due to the criteria we have followed, considering if they were in the same board or if they were related.

Considering those caveats, we have done the following relations: the external temperature and the pressure systems are analyzed together, as are the heating system and the internal temperature system. All other systems are analyzed separately.

We will be focusing on two different validations. On the one hand, we will talk and show all the electronic tests we have performed in order to ensure that the system was working as expected in the design chapter. On the other hand, we will perform the "real tests", the ones recreating a quite similar situation as the one we will have when launching our experiment.

3.1 External temperature and pressure

In this section, we will focus on the external temperature and the pressure systems. As we have already said before, we will first analyze the electronic tests we have performed to prove that the designs where correctly done. After these electronic validations, we will talk about the real tests we have done to check if our objectives could be realized.

3.1.1 Electronic validations

After designing both the external temperature and pressure systems, which were the ones we started with, we bought all the electronic components we needed to build the systems. These systems were built into a Protoboard to check if the values we calculated when designing them with the simulation software Proteus were the same or not.

The first one we built in the Protoboard was the external temperature system (see Figure 3.1). Remember from the design process that when having an output voltage from the external temperature system of 0V, this meant we were at -65ºC, on the other extreme, when the output voltage was 1.1V the temperature was 35ºC.

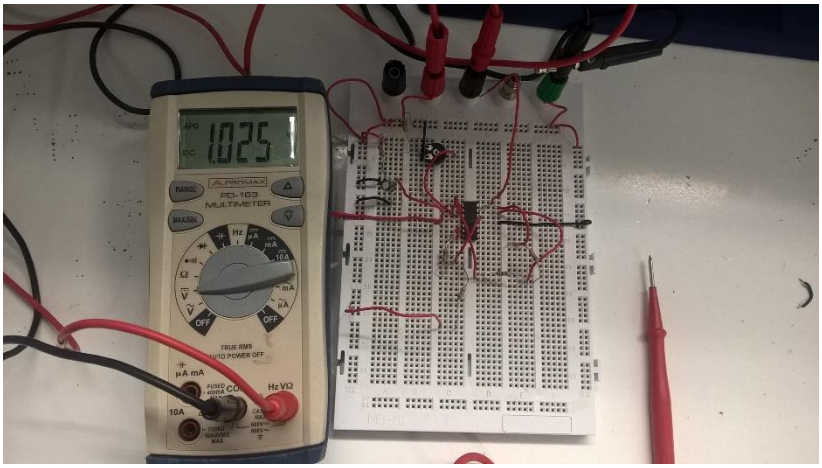


Figure 3.1 External temperature system validation.

In this case, we did the validation in the laboratory. Our output voltage measured with the tester gave us a value of 1.025V; this meant we were at an approximate temperature of 26ºC. We measured the temperature in different situations and in all of them we had values that corresponded to the external temperature. Thus, system worked correctly.

On the other hand, we had the pressure system (see Figure 3.2), which had a range going from 0V to 1.1V corresponding to its output voltage. If the output voltage in the pressure system was 1.1V this would correspond to a pressure of 118.5kPa, but if the value was 0V the pressure would be 0kPa.

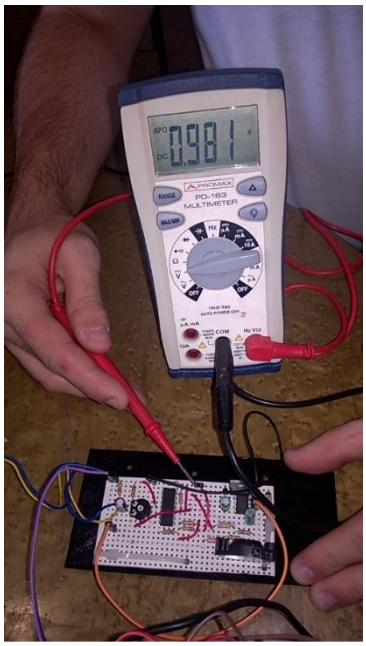


Figure 3.2 Pressure system validation.

We can appreciate that the value is inside the range we wanted, as would indicate a pressure of 105.7 KPa. As for the external temperature system, we made different measurements to see if it really worked. In this case, we used a straw to expand and compress the air going into the pressure system in order to have different values as output voltage.

After testing that the initial designs were correct, we welded both systems in the same electronic board and performed the same series of measurements as we did before. In this case, we had more issues because tin was a big problem. We were not able to ensure the connection between components and every time we reworked it we melted and touched other connections. The first measurements were incorrect as they showed saturated values. Once we made the corresponding improvements all the measurements were the same as the ones we had described in the design process (see Figure 3.3).

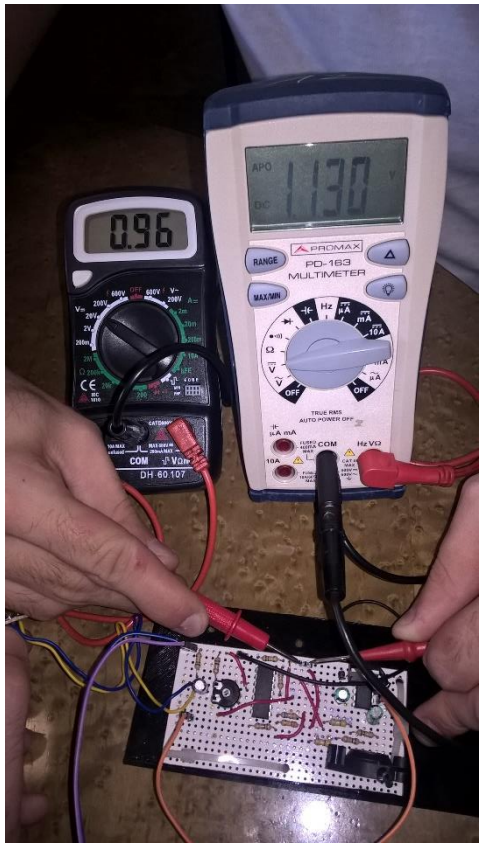


Figure 3.3 External temperature and pressure validation

When we measured the output values in the Protoboard we were not using the microcontroller. Nonetheless, when these systems were built in the electronic boards, we introduced the program we have coded in the microcontroller in order to have the corresponding temperatures and pressures when the measurements were done. After doing so, the results of the tests kept giving correct pressure and temperature values.

3.1.2 Test

When talking about real tests we are referring to the moment when everything is assembled in the probe and we recreate the situations we will have once we launch the probe.

To check if the external temperature sensor and the whole electronics will work at low temperatures, we decided to introduce our experiment in a freezer. The freezer we used was at a temperature of -20ºC. This temperature will be almost reached at 5500m or 18000ft. As our experiment will fly above this altitude and therefore the temperature will decrease until almost -60ºC, this test would not recreate exactly the real situation, however, we will have a closer approach. Our external temperature sensor can read until -65ºC, so it should work correctly in this situation. We did 3 tests in the freezer (view Annex C). In all of these tests we did not reached temperatures below -18ºC because of the power of the freezer, but the sensor kept giving the temperature without any problem. This is reason enough to estimate correct functionality in even lower temperatures.

When we perform our launch, the external temperature sensor will be placed outside the probe, so that it will directly read the external temperature. However, when we did the tests, the temperature sensor was placed inside the structure of the experiment, attached to the upper cover. We did place it there because we wanted to measure the temperature descent we had with the thermal insulation [38]. We were not able to measure actual heat transfer rate because we did not have the thermal conductivity of the material. However, we can compute the mean cooling rate using empirical data from the tests, this will allow us to estimate the quality of the insulation material. We have to take the temperature read by the external sensor at a given time and compare it with the seam measure 60 minutes later. If we do this calculation, we found that the cooling rate is -28.74ºC/hour at a constant environment of -20ºC.

Depending on the type of thermal insulation we use and the thickness of it, we will have a smaller rate, which will be the best for us. This will mean that the inside temperature will not drop dramatically and it might never be equal to the external temperature in the expected mission duration. Considering that there are some parts of the electronics that cannot be at temperatures below 0ºC, it is essential protecting the inside. Figure 3.4 shows the moment when we placed the system inside the freezer. It is important to notice that we cannot be aware of the effect of humidity inside the freezer, nor of the effect of high atmospheric pressure (as compared to that going to be found during the actual mission).

Figure 3.4 Experiment placed in the freezer (first test).

The pressure sensor needs another type of test to prove if everything is working the way we have planned. Our first idea was to place it outside the probe, but we realized it would not be as easy as we thought. Considering that the inside will not be pressurized, we could have our sensor inside the probe and it would work similarly than if we had it outside, although small deviations might take place respect outside pressure

To test the pressure sensor, we had to vary the pressure around the sensor. Depending on the mass flow we have it will read one value or another. The best way to do this test is by using a digital barometer as a reference change the pressure extracting air and measure the amount of air extracted with the barometer. By changing the pressure with the barometer, we can recreate low pressure similar to the ones we will have when our probe climbs to the 35km barrier. For instance, the only measurements we have varying pressure are the ones mentioned above with a straw. Despite we see some changes in the output voltage we cannot exactly know which was the pressure we were exerting to the pressure sensor. We could not find any suitable pressure measurement equipment in the campus facilities, so the pressure system validation is not completed at all, but serves as a good indicator of the behavior of the system.

3.2 Power supply system

The power supply system had always been a headache to us because although the design process was well calculated we always had this feeling that we did not know what would happen when launching the experiment. How long would the batteries last when the temperature drops below 0ºC? Would they last enough to cover the ascent climb before free falling? What we did know was that if the batteries dropped below 6V, the microcontroller would stop working and consequently all the electronics would do so and we would not receive the telemetry.

Testing the external temperature system in the freezer would also be a challenge for the power supply system as we would be able to see if batteries can resist these conditions. It is important to say that introducing the probe in the freezer will mean that the temperature will be constant during the whole time, which is not the same situation we will have when we launch it. The temperature will be dropping following the ISA (International Standard Atmosphere) standards until it reaches the 11km barrier when it will be maintained constant at approximately -60ºC up to 20km altitude. Hence, the behavior of the whole system cannot be measured, we cannot know how would react the inside components with these variations.

The tests we have done revealed important information that will affect our further decisions. In all of the 3 tests, the probe stopped working 1h30’ after turning on the batteries. At that time, the external temperature reached the -15ºC mentioned above. It was surprising for us that this was the most accurate parameter we had extracted from the 3 tests, because it was the same on all 3 situations. External temperature was not exactly -15ºC on all 3 tests, internal temperature was not the same either, but the duration of the batteries was exactly the same. We

revised their datasheet and saw that the operating temperature went from -18ºC up to 55ºC. Therefore, we understood that 1h30' after turning on the probe, batteries already reached the same temperature than the external temperature sensor and they stopped working because they were to near the lower limit of their operating temperature. Hopefully, these tests were performed with enough margin of time before launching the probe and we were able to look for an alternative solution to the battery system.

We only had one issue. The new batteries had to fit in the small volume we had dedicated to the older batteries. This meant that the new batteries should be the same shape or similar. We found other 9V batteries designed by the same manufacturer, having the same shape and weighting 10g less. The main difference was the chemical system, instead of having zinc they were based on lithium [39] (view Annex A). This improved a little bit their capacity and what was more important, the operating temperature, they now reached -40ºC (see Figure 3.5). Another improvement obtained with these batteries is their autonomy, as their capacity is increased with respect to the original ones.

$$Autonomy = \frac{Battery\ Capacity}{Total\ CanSat\ capacity} = \frac{800mAh}{215.9mA} = \mathbf{3.7\ h} \qquad \textbf{(3.1)}$$

Figure 3.5 New Lithium 9V battery.

Furthermore, to see if they could handle the freezer test and pass the -20ºC challenge, we had to do another test. The forth test we did was exclusively done with the lithium batteries, with 2 of them connected in parallel. We have already said that these batteries could work at an operating temperature of -40ºC, this meant that in principle the freezer test should be a normal test. The test went extremely well because we no longer had a 1h30 lifetime. We had the probe inside the freezer for more than 3 hours and we had no problems at all. All systems were working perfectly and batteries were still giving the maximum power.

3.3 Internal temperature and heating system

These two systems are separate ones but they depend on each other. The internal temperature sensor reads values and depending on the temperature we have the heating system will be activated or not.

3.3.1 Electronic validations

Once we had the design process, we wanted to see if these two systems worked together the way we had expected. In this case, the electronics we had to build were very easy because they only comprised 4 elements: the 2 resistors, the MOSFET and the internal temperature system.

Just as we did with the external temperature and pressure systems, we first connected them to a Protoboard. Our first try had not the aluminum plate, as we only wanted to see if the two resistors worked correctly. After this attempt, we saw that the heat radiated by the 2 resistors was not spread across the whole electronics inside the FourCast, instead if concentrated locally around the resistors; that is the reason why we added the aluminum plate. Once we had both resistors equipped with a single aluminum plate that packed them as a single unit, we did a second attempt (see Figure 3.6). Therefore, to test both systems we connected the microcontroller to settle the transistor as a switch. The value selected was 2ºC, if the internal temperature was below it the transistor would automatically led the current pass and the resistors will heat up the electronics. We took an ice cube to force the internal temperature sensor to read lectures below 2ºC. We saw that when the temperature dropped, the heating system automatically heated up, proving that it worked as planned. The following condition we gave to the transistor was that if the value read was above 3ºC, it should turn off. We use these two values in order to prevent unnecessary flickering that could damage the MOSFET.

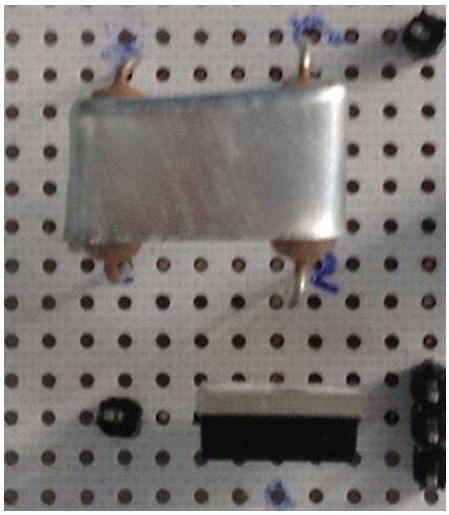


Figure 3.6 Heating system validation.

3.3.2 Test

The electronic validation was successfully executed. As we did with the other systems, we wanted to check how was their behavior when recreating an environment similar to the one we would have when launching the actual device.

To test these systems, we also appealed to the freezer test because it is the one that place all systems in a similar environment as the one found when we launch the probe. The internal configuration we have decided for FourCast had a high importance because the heating system had to be in front of the electronics and specifically in front of the amplifier because it is the most delicate part.

The internal temperature sensor was not attached to the cover like the external temperature sensor; instead it was in front of the recovery system, to use the synergy given by the slight warming of the voltage regulator that heated a little bit too. This information is quite useful because when we look at the telemetry we had in all the tests, we can appreciate quite notable differences between both temperature sensors. When analyzing the cooling rate of the internal sensor we find a rate of -15ºC/hour, as compared to the external one of -23ºC/hour.

At the beginning, the internal temperature sensor measurements were quite constant for the first 30 minutes of the test; during this period of time the temperature had very small variations and it continued to have high values. As time progressed, the temperature started to drop. However, due the thermal insulation layer, it did not drop as fast as it did with the external temperature. At the end of the 3 tests, the internal temperature had not the same lecture on all 3, yet it was very different from the external temperature. The values were never under 6ºC, meaning this meant that the transistor continued to be switched off. This was very positive because the electronics resist all the tests, without having to appeal to the heating system.

Furthermore, in the fourth test we did the time inside the freezer was longer, and we were able to see how the heating system worked. When we passed the 2 hours' barrier was overcome, we started to see values in the internal temperature sensor that were below 2ºC. This meant that the heater had been activated by these low temperatures. After 30 seconds, when we had the following reading the system already was above 3ºC and so the heating system was shut down again. It kept going this way during 1 hour and the heating system handled all these changes perfectly. We could appreciate how it was warming up all the electronics inside the experiment because the temperature increased when the heating system was turned on. In order to see if the transistor was activated or not we create a Boolean what in the Arduino code to show us its activity (view Annex C).

As we have indicated, it is not only the configuration inside the probe that will help avoiding close to 0ºC temperature but also all the thermal insulation materials that we have used outside and inside the probe (see Figure 3.7). We have introduced a layer of aluminum between both cylinders, then we have reinforced both covers with some insulating tape, and after this we added another layer of

aluminum outside the bigger cylinder. Finally, we reinforced the aluminum with another layer of insulating tape (see Figure 3.8).

Figure 3.7 Aluminum and insulation tape.

Figure 3.8 Design of the external thermal insulating materials.

3.4 XBee communication system test validation

Beside the analog sensor systems, the digital systems that communicate with the microcontroller had also been tested and validated. There are some requirements that must be fulfilled in order to guarantee operability for the whole project. The XBee communication system is the primary communications system of the probe, and its functionality needs to be assured as the secondary communications system (that uses mobile 3G signal) might stop working if the signal is lost.

The requirements for the primary communications system are the following:

- To receive all the packages sent by Arduino.

- To be able to access the content of those packets.
- To be able to work when the antenna is not directly pointed to the probe.
- To have a range equal or above 35km.

3.4.1 Test design

In order to verify if the communications components that we chose achieved these requirements, a test had to be designed, trying to mimic the conditions of the real operation. In our case, most of the communications parameters are already fixed, as described in Chapter 1. Then, there are only two main things that need to be tested. First, we need to make sure that we can receive data from Arduino in the ground station, using the X-CTU software. Then, we need to verify if the same data can be received at 35km distance, in line of sight.

We had to find two locations, separated 35km approximately with direct line of sight contact, like two mountain peaks. We found two ideal locations nearby, the Collserola Tower and the Monastery of Montserrat (see Figure 51).

Figure 3.9 Collserola to Montserrat line of sight distance

The configuration of the XBee modules and Arduino code will be the same as the day of the launch, with the following values:

- **Transmit Power**: 315mW
- **Baud rate**: 9600 bauds

3.4.2 Test

The test was conducted the Saturday 24th of September. The test started at 08:00 and the duration was approximately 2 hours.

First, one of us arrived to the feet of the Collserola Tower, and set there the ground station, connecting the XBee to a laptop using the USB shield, and the

YAGI antenna connected to it. Then, the X-CTU software was configured to record all the received data, and the communication channel was opened. The YAGI antenna was pointed facing Montserrat (see Figure 3.10), the location where the probe was.

Figure 3.10 Antenna position of the test, Montserrat at background

The other member went to Montserrat (see Figure 3.11), and once he was there he connected the probe systems. The data transmission started immediately, but the data packets could not be fully retrieved at the ground station. 8 minutes after the transmission had started, the ground station received its firsts packets, by the sequence number of the packets we could know that were recently generated packets, so the first ones were lost. We also noticed that not all packets were received, as some sequence number would just skip.

Then we started to make request from the ground station to FourCast. We used the RTS (Request To Send) functionality of the X-CTU software. This mode, sends constant pings to all connected modules to the ground station, and the modules answer if they are ready to receive data. We did not have any intention to send data from the ground station to the probe, but this is an excellent method for testing range, as the signal has to travel to the receiver in the probe and then, its reply has to make all the travel way back to the ground station. Once this mode was activated, we successfully received a response for all the RTS sent. This meant that the probe was in range, at 35km.

Figure 3.11 Montserrat, Collserola at the background

We realized that some of the data packets (with probe sensor data) were not received because they were too huge and bit errors appeared. The RTS packets were approximately 80 times lighter and all of them were received successfully. As we do not need to send data at a high rate, we reduced the size of the packets, sending them on a longer time interval. In this way, less bits form the packets and more information can be retrieved, being able to receive all packets as usual.

The conclusions we can extract form this test are that:

- Packet size is important to assure good communication at 35km range. It is important to use small packets.
- Line of sight is also important. If the ground station antenna is not correctly pointed, even the smallest packets are not received.

3.5 GPS positioning system test validation

The GPS system is another of the digital systems that communicate with Arduino using serial communication. As described in the Chapter 1, library *SoftwareSerial* will be used in order to emulate a serial port and stablish communication within the Arduino and the GPS device.

The function of the GPS system is quite simple, positioning. The position requirements for the project are the following:

- Obtain correct values of Latitude, Longitude and Altitude.
- Detect errors in positioning and erase false data.
- Great precision is not needed, but will be positively evaluated.
- Receive data even with the GPS antenna inside the probe.

3.5.1 Test design

In order to design an effective test, let's consider briefly how GPS receivers work, and how they solve the positioning equations. GPS devices solve iteratively the position equations, proposing solutions and minimizing the error, using 4 satellite references. This is why GPS devices do not show the position immediately after they are turned on, but after a few seconds or even minutes. They require some time in order to find an appropriate solution, iteratively. There are ways to reduce the startup time, for example, with more powerful processors for calculations or using more than 4 satellites. The relative position of the satellites also provides additional help in order to calculate the user's position. It is always better to have the satellites placed in wide angles respect the receiver. Nevertheless, the GPS antenna must be a few minutes in a clear position with direct line of sight with the sky, in order to start receiving GPS data. Then, the test will go on as follows:

- First, with the external structure opened, the antenna will be placed on top of the wires and near the ceiling overture. We will have to wait until the

green LED of the receiver blinks indicating that we have correct position data.

- Once we receive data, we will fully close the probe, like we will on the day of the launch. Then we will see if we continue receiving data, even when the antenna is not in direct line of vision with the sky.
- The test must be performed in an open-air location.

3.5.2 Test

Once we powered the device, it took 3 minutes and 53 seconds to start receiving GPS data. The first sentences that we received contained a large amount of error, around 50 meters. After 4 more minutes, the error reduced to 10 meters and the values were stable. The minimum error that we achieved was 10 meters (see Figure 3.12).

Figure 3.12 Test location with received measures.

To sum up, out of this test we draw the following conclusions:

- The GPS of our system needs about 4 minutes to start receiving data, with the ceiling opened and the antenna facing the sky.
- It is important not to close the probe until a few minutes more have passed. If the probe closing is made immediately after the GPS starts receiving data, it exists a high probability of losing coverage.

3.6 FourCast recovery system test validation

The final digital system that needs to be validated is the IoT module, the ESP8266. Let's remind the main functions of this system:

- To communicate telemetry even when the XBee signal is lost (ideal for recovery)
- To insert telemetry into a SQL database which is processed in real time and allows powerful visual representation of the data.

The main advantage of this system is that instead of sending raw data like the XBee it sends data to a server that processes it and creates real measures. Then the measures are inserted into a SQL database in different columns, with the correct data type. This allow us to connect different tools to that database (like Excel) and perform real time analysis. Then, all the mission data is visualized in a powerful way, with rich visuals.

The minimum requirements for this system are numerous, as it is the most complex system of the whole probe and also involves external agents (the remote server with the SQL database) so they will be classified as follows:

- **Wi-Fi Modem:**
  - Obtain signal even when the probe is closed.
  - Recover signal automatically when it loses it.
- **ESP8266:**
  - Connect automatically to the available Wi-Fi network, storing the password.
  - Reconnect to the Wi-Fi network if it loses its signal.
  - To format HTTP GET request correctly using HTTP/1.1
  - Open and close connection sockets properly.
- **Server**:
  - Be able to insert all the experiment received data into the database.
  - Handle correctly data types.
  - Log the errors registered.
- **Analysis Software (Power BI):**
  - Refresh in real time and show results properly.

3.6.1 Test design

Although the requirements are complex, the design of the test is simple. We will connect the modem Wi-Fi and the probe, and we will wait until the HTTP requests start arriving to the server. Then, we will close the container and see if the signal is still good. We will open it again, and we will turn off the Wi-Fi modem. After a while we will turn it on again and see if the HTTP requests resume correctly. Before noting the results, it is important to notice the following:

- *The system is not designed to handle queues. If there is no signal, the HTTP request will be trashed and the data cannot be recovered for that particular transmission. A possible way of improving the system would be adding a buffer and queue managing, using standard TCP features.*

3.6.2 Test

In the first test we made, not all requests arrived to the server. We marked each request with a sequence number (increasing 1 in every request). We noticed that

only 1 out of every 2 requests arrived correctly. Then we deduced that there was a problem with the connection socket. The requests are separated by a time interval of 30 seconds, so we understood that that was long enough for the socket to close automatically. But it seems that the ESP8266 keeps the socket opened for more than 30 seconds, so when the module tries to open it up again in the next interval it fails and the data is not sent. We fixed this issue forcing to close the socket on each iteration.

Then we tested again and it worked flawlessly, all the data transmissions correctly reaching the server. Thereafter, we closed the FourCast and the data was correctly sent too. Then we shut down the Wi-Fi modem, and after a while we restarted it. The ESP8266 automatically did reconnect, and the requests to the server were correctly resumed.

In the server side, all requests could be correctly interpreted. When the GPS does not receive data, dummy values are inserted, easily detectable by the server code, and it inserts they as NULL on the database. The other values are correctly received, and stored with the appropriate data format. This is important for the dates for example, that are stored as SQL date times, meaning that we have millisecond precision.

The Power BI software that analyzes the data correctly detects the data types and recognizes the latitude and longitude as geographical coordinates. The rest of the decimal values (like temperature) are also correctly read. We also checked that the data is not updated in real time, the refresh button of the Power BI toolbar must be pressed, but this is a minor inconvenient. We created a Power BI report like the one in Figure 3.13 that shows all the information we need with powerful visual style. In that particular report, the external pressure system was not working correctly, so the values showed are not correct.

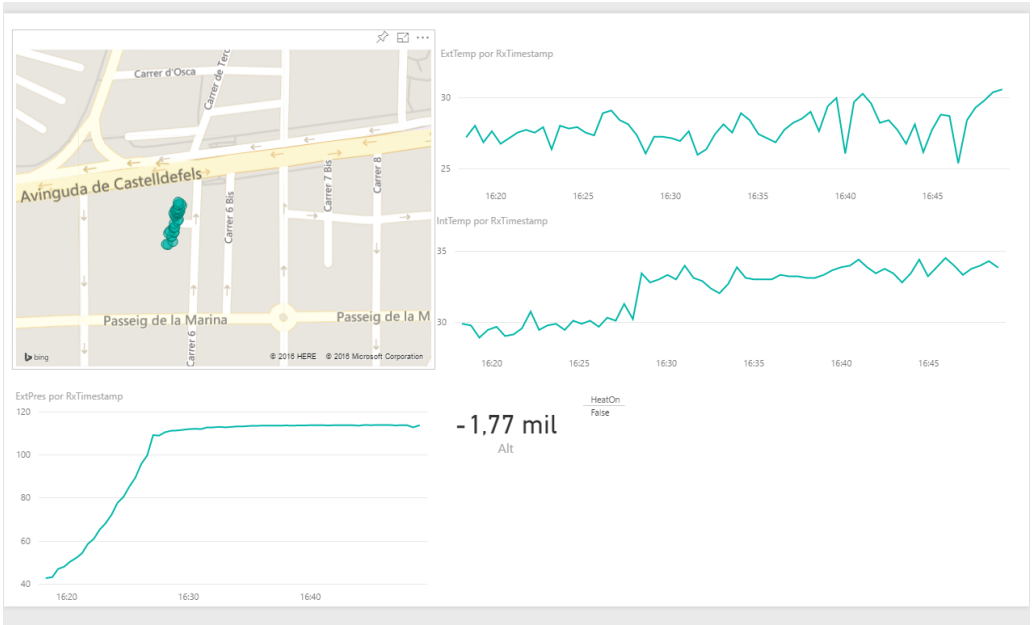


Figure 3.13 Power BI report with real time data

In Figure 3.14 we are able to see the data inserted into the SQL database.

	RxTimestamp	ExtTemp	IntTemp	ExtPres	Lat	Lon	Alt	HeatOn	TxSeq	TxTimestamp
35	2016-09-30 16:35:02.000	27,47	33,01	113,65	41,268745	1,976086	10,4	0	34	2016-09-30 16:34:43.000
36	2016-09-30 16:36:01.000	26,89	33,01	113,77	41,268761	1,976095	10,1	0	36	2016-09-30 16:35:43.000
37	2016-09-30 16:36:31.000	27,77	33,33	113,77	41,268784	1,976105	7,5	0	37	2016-09-30 16:36:13.000
38	2016-09-30 16:37:01.000	28,26	33,23	113,77	41,268803	1,976123	1,8	0	38	2016-09-30 16:36:43.000
39	2016-09-30 16:37:31.000	28,55	33,23	113,77	41,26881	1,976139	3,2	0	39	2016-09-30 16:37:13.000
40	2016-09-30 16:38:01.000	29,04	33,12	113,88	41,268845	1,976166	7,2	0	40	2016-09-30 16:37:42.000
41	2016-09-30 16:38:30.000	27,67	33,12	113,77	41,268791	1,976143	6,3	0	41	2016-09-30 16:38:12.000
42	2016-09-30 16:39:00.000	29,43	33,33	113,88	41,268772	1,97613	5,3	0	42	2016-09-30 16:38:42.000
43	2016-09-30 16:39:30.000	30,01	33,66	113,88	41,268761	1,976119	3,2	0	43	2016-09-30 16:39:12.000
44	2016-09-30 16:40:00.000	26,1	33,87	113,99	41,268776	1,97611	3,9	0	44	2016-09-30 16:39:42.000
45	2016-09-30 16:40:30.000	29,72	33,98	113,99	41,268776	1,976109	5,5	0	45	2016-09-30 16:40:12.000
46	2016-09-30 16:41:00.000	30,31	34,41	113,99	41,268818	1,976115	2,3	0	46	2016-09-30 16:40:41.000
47	2016-09-30 16:41:29.000	29,62	33,87	113,88	41,268799	1,976079	0,3	0	47	2016-09-30 16:41:11.000
48	2016-09-30 16:41:59.000	28,26	33,44	113,99	41,268768	1,976061	1,8	0	48	2016-09-30 16:41:41.000
49	2016-09-30 16:42:29.000	28,45	33,76	113,99	41,268738	1,976075	6,7	0	49	2016-09-30 16:42:11.000
50	2016-09-30 16:42:59.000	27,77	33,44	113,99	41,26862	1,976055	17,3	0	50	2016-09-30 16:42:41.000
51	2016-09-30 16:43:29.000	26,79	32,8	113,99	41,268574	1,976064	24,4	0	51	2016-09-30 16:43:10.000
52	2016-09-30 16:43:59.000	28,16	33,44	113,77	41,268627	1,976076	17,8	0	52	2016-09-30 16:43:40.000
53	2016-09-30 16:44:28.000	26,2	34,41	114,11	41,268532	1,97603	21,3	0	53	2016-09-30 16:44:10.000
54	2016-09-30 16:44:58.000	27,77	33,23	113,99	41,268536	1,976033	22,5	0	54	2016-09-30 16:44:40.000
55	2016-09-30 16:45:28.000	28,84	33,87	114,11	41,268581	1,976066	22,7	0	55	2016-09-30 16:45:10.000
56	2016-09-30 16:45:58.000	28,74	34,52	114,11	41,268665	1,976066	19,2	0	56	2016-09-30 16:45:40.000
57	2016-09-30 16:46:28.000	25,42	33,98	114,11	41,268566	1,97599	17,1	0	57	2016-09-30 16:46:09.000
58	2016-09-30 16:46:58.000	28,45	33,33	113,88	41,268482	1,976046	30,1	0	58	2016-09-30 16:46:39.000
59	2016-09-30 16:47:27.000	29,33	33,76	113,99	41,268627	1,976062	18,9	0	59	2016-09-30 16:47:09.000
60	2016-09-30 16:47:57.000	29,82	33,98	113,99	41,268589	1,976009	18,5	0	60	2016-09-30 16:47:39.000
61	2016-09-30 16:48:27.000	30,41	34,3	112,98	41,268456	1,975961	29,8	0	61	2016-09-30 16:48:09.000
62	2016-09-30 16:48:57.000	30,6	33,87	113,88	41,268452	1,975983	29,1	0	62	2016-09-30 16:48:39.000

Figure 3.14 Test data sent by the Arduino into the SQL Database

The conclusions we got after there are the following:

- The system works as expected, all the requests are received and can be visualized in real time.
- It is important to close the socket after each request, and reopen it again.

3.7 Parachute

This system is one of the easiest systems to validate; because we do not depend on coding or electronic connections it is entirely mechanical. We already know which type of parachute we will use in our project; it will be a small umbrella with a heptagonal shape and a radius of 0.53m.

Once we bought the umbrella we had to get rid of the metallic elements that hold each vertex and the central bar leaving only the fabric itself. To complete the fabrication process, we had to do holes on each vertex in order to pass a rope on everyone. Each rope measured the same length, 120cm. We did not want to have the parachute just on top the probe. In addition, we thought that having this separation would be beneficial for its further deployment.

Therefore, we took each rope and connect it to a carabiner that was holding the system. Once the parachute was built we wanted to test it to see if it really worked and if the impact with the ground was not that strong. In order to test its

functionality, we looked for an object (see Figure 3.15), which weighted approximately the same than the probe (700gr). Our first tests were not directly done with the probe just in case calculations were not done correctly and the probe could have broken. We performed several releases from an approximately height of 15m. All of them worked perfectly proving that the selection of the parachute was the correct one (see Figure 3.16).

Figure 3.15 Parachute with an object.

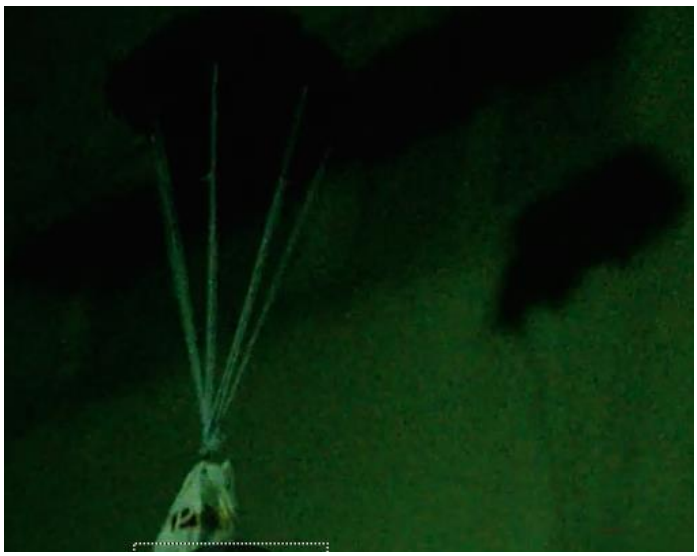


Figure 3.16 Parachute test with a 700gr payload.

We saw that the impact was not that strong, our probe would handle it without any problem. The impact's strength is very important due to safety reasons. When we talk about the crash, electronics are almost the least important thing. We do not want to have a strong crash because we could seriously hurt somebody.

Keeping these safety principles in mind, we were worried about the quality of cloth materials of the umbrella. We were worried because it looked fragile and could be ripped off easily. This is the reason why we decided to buy a more professional parachute. Luckily, we found an ideal one, in the Stratoflights [40] website (see Figure 3.17), with the same dimensions as the initially proposed umbrella. We performed the same tests on the new parachute and, as expected, the results were the same as with the umbrella.

Figure 3.17 The new parachute

CHAPTER 4. LAUNCH PREPARATION

In this Chapter we describe all the aspects involved in the preparation of our probe launch. We discuss the materials needed to fir the launch and the expected landing of the probe and describe the protocol and appropriate documentation necessary to comply with the existing regulations. Finally, we explain the expected results and the actual output from our experiment.

4.1 Finding the appropriate location

The first part in the launch preparation is analyzing the appropriate location where we should execute our experiment. When selecting this location we have to study many aspects such as meteorological conditions or proximity to important airports and highly populated areas.

First of all, it is not possible to launch our experiment from the Campus where the EETAC is located. It is too close to the airport, and there are many different airways crossing this area, as aircrafts are approaching Barcelona Airport (LEBL). Actually, regulations do not allow this kind of activities at less than 8 km from an airport [41]. In addition, we cannot be close to a crowed area in order to avoid possible problems when the satellite lands.

Once we have this information, we had to discard all the Baix Llobregat area and Barcelona surroundings. So, we had to look for an aerodrome far enough from Barcelona and with reasonably good expectations of good meteorological conditions. We already know that our probe might ascend up to about 35km. At this altitude, winds in the Catalonia area can displace the satellite up to 200 km in the horizontal direction (normally North-East) heading. This information is very useful because we do not want them. If it lands in the sea, communications will stop working and it would almost be impossible to recover it.

As a first option, we thought that Los Monegros, near Zaragoza might be a good spot to launch our probe, because it is barely populated and far from the see. However, the strong winds characteristic of this area might jeopardize the recovery of the probe. Therefore, we discarded this option.

We then checked the area of Catalonia. We found many aerodromes, as the ones located in Mollerusa or Igualada. We contacted both of them and we only received an answer from the aerodrome in Mollerusa, which is coordinated and directed by Jimmy Capell. After checking the charts, we realized that Mollerusa might be a good option (see Figure 4.1). We looked at the specific location of the aerodrome and saw that it fulfilled all the requirements. It is far from a crowded area like Lleida. Although, the closest point to the sea is 60km away in straight line, it is headed South-East, and thus, not in the direction of the dominant winds (North-East). Finally, the weather forecast during the week in which we could launch the probe was quite good; winds were not above 10km/h, most of the days where sunny and there was low chance of rain.

Figure 4.1 Mollerussa location

4.2 Administrative requirements

The standard protocol to launch a probe like ours involves applying for a specific permit: NOTAM (Notice to Airmen). It must be requested in order to inform that a specific activity will be executed in a selected area of air space, in case some changes had to be applied to air traffic.

A NOTAM is requested to the corresponding authority of the country in which the launch is performed (ENAIRE in our case), and it takes about 3 weeks to be issued. The application form (see Annex D) requests information such as dates of the launch, duration and start time of the activity, location with its coordinates, balloon characteristics and the petitioner and organizer signatures.

Once the NOTAM is obtained confirmation of the corresponding activity must be communicated the day in which the experiment is done, both before it starts and once it is concluded. Figure 4.2 shows the NOTAM publication.

(D2940/16 NOTAMN
Q)LECB/QWLLW/IV/M /W /000/999/4137N00051E006
A)LECB B)1610290700 C)1610291100
E)ASCENT OF FREE BALLOON (SOUNDING) WILL TAKE PLACE WI 10KM
RADIUS OF 413641N 0005118E. MOLLERUSA
BALLOON FEATURES:
TYPE: LIGHT
COLOUR: WHITE
DIAMETER: 3M
TOTAL WEIGHT/SOUNDING WEIGHT: 1.2/0.7KG
ASCENT SPEED: 4.7M/S
F)SFC G)UNL)

Figure 4.2 NOTAM publication

4.3 Materials needed and expected results

Once we had selected the location and we had the NOTAM publication, we were able to launch the probe. In order to launch it, we needed 2 additional elements: a Helium balloon and a Helium tank.

We looked for a professional balloon in order to satisfy all the requirements to lift the satellite. The website called Stratoflights in Germany provided us with a convenient balloon for our experiment. As our all probe plus the parachute weighted approximately 800 g, we decided to purchase the 1600 g balloon. Additional characteristics were the following: it had 2 m diameter, and it could ascend up to 36km with a climb rate of 5m/s. To fill this balloon, we needed $4m^3$ of He. Buying He is not cheap because there is shortage of it all over to the world. Furthermore, He is sold in big quantities. The bottle we have bought has $9m^3$ of He, more than twice the amount of He we needed.

The balloon must be filled just before we launch the satellite. Once we have the balloon filled we l used a polyethylene cord to connect it to the parachute. Thereby, when the balloon reaches its ceiling altitude and explodes the parachute will be deployed without having to use any electronical or mechanical system to activate it. Furthermore, we had also included a RADAR reflector (see Figure 4.3) just below the probe in order to make the probe detectable by all primary surveillance RADARS. To accomplish all the steps previous to the launch, we did a checklist in order to control everything was working before the launch (view Annex E).

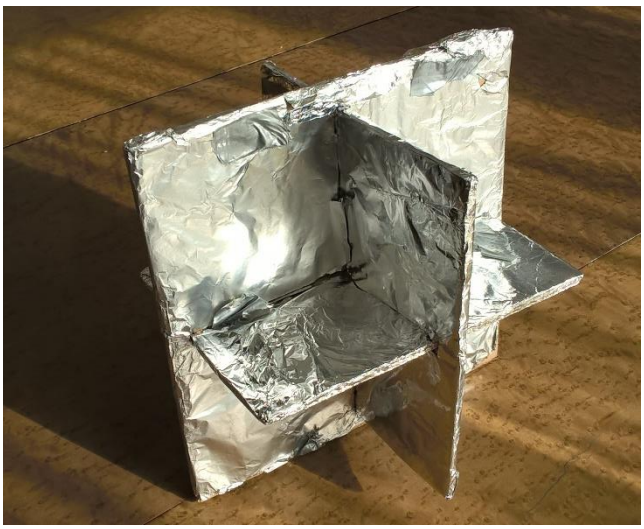


Figure 4.3 RADAR reflector

Having carefully chosen the location and day of the launch, and after the number of tests which we had performed, we expected the experiment would develop without problems. Nevertheless, we were worried that the tests we had done recreating low temperatures (-20ºC) might not be close enough to the actual temperatures expected during the ascent of the probe (-60ºC). We therefore got additional insulation materials from a local manufacturer to protect the probe.

In relation to the behavior of our probe before very low temperatures there were two key factors: the time it would take the balloon to ascend until 36km and the effect of humidity to the probe. A fast climb would represent a shorter exposition to low temperatures, and low humidity might help by affecting thermal conductivity. Finally, we expected that both the GPS and the retrieval system

could work throughout the experiment. We knew that GPS would stop working above 18km and that WIFI coverage would be also turned off at some altitude. However, when the probe starts falling and enters the 18km barrier both systems should work again.

4.4 FourCast launch on November 19th 2016

Although all the launch preparations were done, we were not able to launch the probe on October 29th as planned, due a problem with the supplier of the balloon. Nevertheless, we could manage to obtain permission from ENAIRE to launch it again on November 19th, at the same location that we planned originally. The meteorological conditions that day were ideal also for the launch, and all the materials needed were already in our hands days before the chosen day.

We arrived to the Mollerussa aerodrome at 08:00 AM, with all the materials and tools needed, including the helium gas. In order to inflate the balloon, we followed up the instructions provided by the manufacturer. In addition, we added a plasticized note with contact information in case that we could not estimate landing position pf the probe and somebody else finds it, as can be seen on figure 4.5.

Figure 4.4 Note attached to the probe.

When the balloon was properly inflated we proceeded to release it, but before we checked up that all systems were working correctly. Figure 4.5 shows the previous moments between the launch. The launch took place at 09:46:38 AM.

Figure 4.5 Photography taken minutes before the launch

The firsts moments of the experiment went according to the planning. The probe ascension rate was 5m/s, exactly as planned. We received data from the two communications systems, and were close to the expected values. The only system that didn't work was the external temperature sensor, as we always received the same dummy and impossible value, 35ºC. We estimate that this failure could be due a cable disconnection, maybe because the winds or the initial lift force. We could follow up all the telemetry of the probe for 28 minutes in the IoT system, until it reached an altitude of 5000m. At that point, the Vodafone Mifi modem could not connect to the Internet, because 3G mobile signal was not available. We still were able to receive data using the primary communications system, the XBee Data Link for the next 8 minutes, when the probe reached an altitude of 7300m meaning that we have actual data for a total of 36 minutes.

The trajectory that we could follow up in real time using the data inserted into the SQL database by the IoT system can be seen in figure 4.6. We lost communication when the probe was above Preixana.

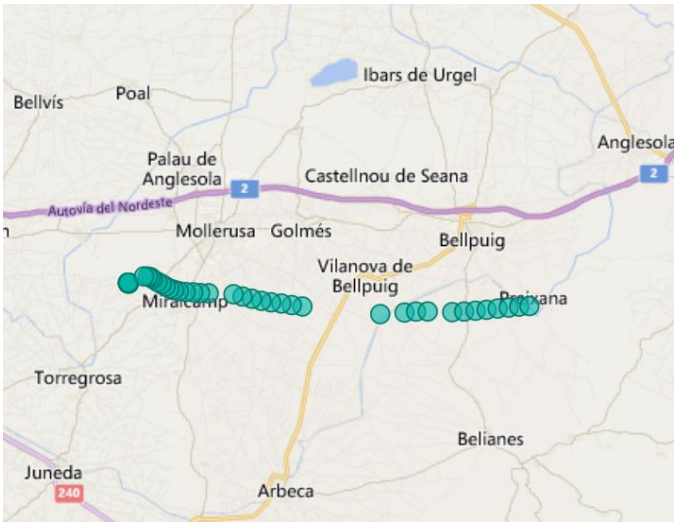


Figure 4.6 Trajectory of the experiment during the firsts 28 minutes.

Also, we present the pressure data obtained during the first 5km of the lift of the experiment, as a function of the altitude registered by the GPS. It can be seen on figure 4.7, as blue dots, compared to the International Standard Atmosphere data, showed as an orange line.

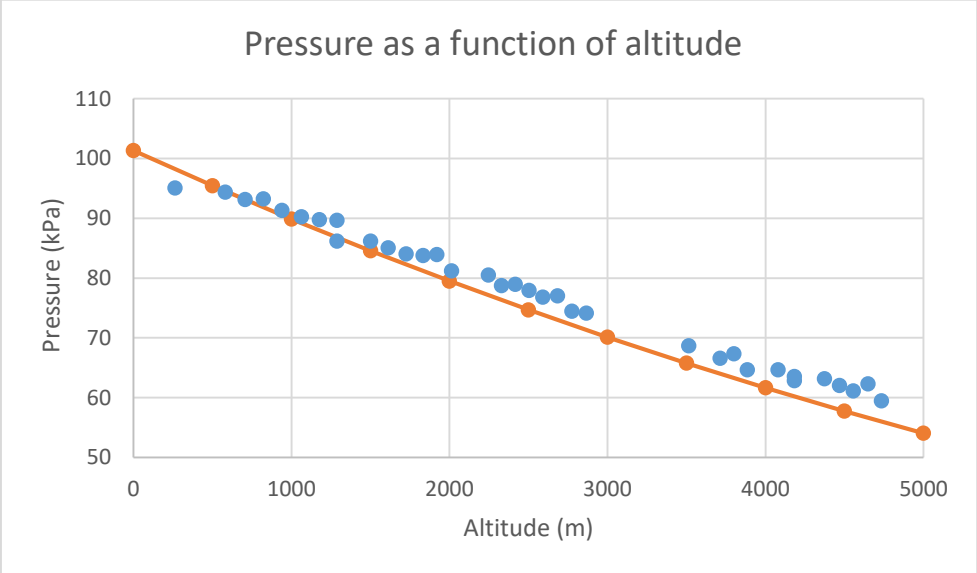


Figure 4.7 Pressure measurements

Besides the external temperature sensor, all the other systems worked flawlessly. We could not see the heating system in action, as temperature inside the probe was always above 20ºC during all the experiment phase where we had contact, as shows figure 4.8. The first 45 minutes the probe was closed up, but at the ground. Temperature increased once we closed up the probe, as the electronics inside it produced heat and the external temperature wasn't cold enough. Even after being launched, internal temperature continued increasing during all the phase when we had contact.

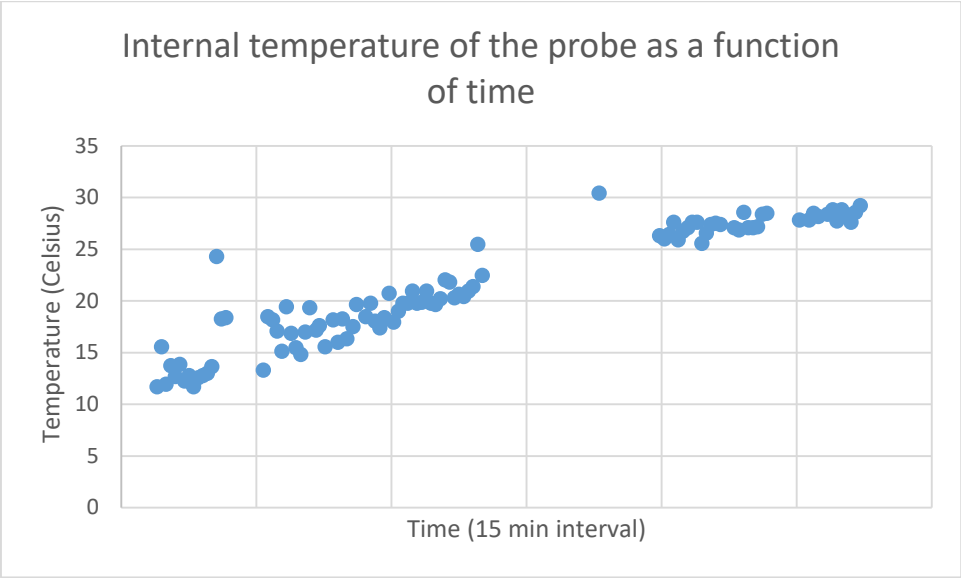


Figure 4.8 Internal temperature of the probe

We estimate that the main cause of the communications breakdown for the case of the primary communications system was due the strong lateral winds, that deviated the probe a significant distance from its vertical ascension path and soon was out of the range specified by the manufacturer. We also didn't receive any signal anymore from the IoT communications system, that supposedly should have activated during the descent of the probe, after the balloon exploded. We believe that the probe ran out of batteries, because the amount of helium inside the balloon was not enough. This could have probably caused that the probe didn't explode as planned, and spend a long time gliding around an altitude of 36km.

CHAPTER 5. CONCLUSIONS

In this Chapter, we derive the main conclusions from our project. Additionally, we suggest possible improvements which might eventually be implemented in a future similar work. In doing so we take into account the underperformance cases, problems or failures that we experienced during the design and construction of the experiment.

We designed and built an 800g meteorological probe able to measure and send to a ground station atmospheric temperature and pressure values from a few km heights. Additionally, a GPS system allowed the determination of its precise position and, eventually, the probe's recovery. We used a helium balloon to launch our probe at Mollerussa aerodrome, after obtaining the required formal permits (NOTAM), and choosing the exact date according to the local meteorological conditions. The probe actually reached a 7.6km altitude and was able to obtain and send external pressure data, whereas some problem with the external temperature sensor (probably an unexpected disconnection) yielded unrealistic values from the time of the launch. Pressure and position data were received since the launch and during the next 36 minutes. At that time communication with the probe was lost, probably because the probe went out of the expected range, due strong winds in the higher layers of the troposphere. This fact, unfortunately, made recovery impossible.

5.1 Main issues during the design and construction processes

From the point of view of the design and construction energy supply has been the main technical problem during the whole construction and design phases. The natural environment for a probe like this is the upper layers of the atmosphere, so there is no way to plug a power line into it. This means that power is a limited resource and so all electronics onboard must be designed keeping that in mind. As a consequence, it was more convenient to use high-end electronics, despite its price. All digital devices should use 3.3V (in order to reduce power consumption) and all analog devices should be as low voltage as possible. It was also important to use high value resistors in the design of the conditioning circuit for the analog sensors, keeping in mind that this also increased thermal noise. Moreover, no power cables should be kept free because, if due to the probe's motion, a negative line contacted a positive, line the battery might drain in a matter of minutes, and overheat all the electronics. As a summary, power in the probe was a very limited resource and its consumption had to be minimized in order to ensure endurance.

Another major difficulty was the thermal conditioning of the whole experiment. It is important to choose components that have low operational temperatures, even if they are relatively expensive. In our case, the sensors have good operational temperature ranges, but some electronics must be kept always above 0 degrees, so a heating system was needed. Temperature changes can make that some electronics perform better or worse, for example, batteries dry quicker in cold environments but sensors have less noise. Heat flow can be optimized in several ways, and we found out that the most effective way was to add layers of insulating

material, in the exterior of the probe, and use thermal resistant tape to join the pieces. We realized that aluminum composites were light and have good insulation properties.

Before printing the 3D structure and placing all components into PCB boards, it was important to perform multiple Protoboard tests in order to avoid problems such as saturation of active elements. If these electronic validations were correct, errors occurring after the assembly are probably at a software level and can be fixed after debugging the corresponding code. It is also important to keep in mind that all the systems must work together and coexist, so correct individual device behaviors do not ensure a good performance of the entire system.

In order to maximize the possibilities of receiving the data from the sensors we designed two independent systems of data transmission, using independent and coexistent methods. This was expected to be highly useful during the launch of the probe. In order to decide which systems must be redundant we have to keep in mind several factors, such as the individual probability of failure and the importance of the system in terms of achieving the mission goals. It is important to conduct FTA (Failure Tree Analysis) trying to quantify the failure probabilities. We decided to add redundancy for the communication subsystem because if it failed we would be unable to obtain data from any other system. We also quantified the probability of failure of this subsystem as moderate-high, because it could fail on multiple factors (antenna disconnection, wrong facing on the antenna, etc.).

We found that planning was very important in this project if we were to meet the desired deadline. Gantt diagrams (view Annex F) and hour journals have proved useful tools. It was very important to stablish a master planning of the whole project, and divide it into phases each one with a deadline, but including reasonable safety margins.

In our case, at the beginning of each week we planned and, if necessary, reschedule the next 14 days. We also learnt that is quite important to order the materials with enough time, and do not rely 100% on the shipping times estimated by the vendor.

Another main concern during this project was to keep a low budget. We found several vendors for the materials we needed. In electronics, prices can have huge variations, depending where you buy it, and the manufacturer. This is why alternatives were always considered and multiple vendors compared. In some cases it was mandatory to buy a high-end product (such as the XBee PRO, due the range requirements) but in other cases lower quality (and cheaper) products could be used. This optimized the cost of the whole project (view Annex G). It must be noted that this kind of projects are not cheap because of the extreme conditions at which the electronics must be able to operate and communicate require many high-end devices.

This technical memory itself is expected to serve as a comprehensive support document for students who might want to recreate the experiment, alter it or improve it. It was written emphasizing the modular nature of the project and

carefully describing all the materials needed and the steps necessary to complete it. It was also interesting to realize the wide variety of information from different courses of the degree which we have implemented to develop this project. It allowed us to use creatively and put into practice theoretical information and has helped us to consolidate our knowledge. (view Annex H).

5.2 Possible improvements and future plans

After the additional experience we have achieved after developing the present work we consider there are some aspects which might be improved. In particular, it might be helpful if future students decide to build a similar experiment.

Electronical improvements: Noise might be reduced or alternative components might be considered, depending on the actual goals of future missions. For instance, smaller devices should be considered if an actual CanSat is to be built.

- **Additional systems:** We included low power consuming devices necessary to characterize atmospheric pressure and temperature. A variation of the project could include additional systems, such as a radiation sensor or a HD camera.

Alternative ascension method: A variation of the project could use a rocket instead of a helium balloon. This would provide additional workload on the student, such as applying thrust control and stability design, or adapting the dimensions of the probe.

Different materials: Other types of materials, instead of PLA plastic could be used to improve mechanical and thermal properties.

“To infinity and beyond” – Buzz Lightyear

“Planet Earth is blue, and there is nothing I can do” – David Bowie

CHAPTER 6. REFERENCES

[1] T. Yasaka, "University Hands-On Space Education -Japanese Case-," UNISEC, 1998.

[2] D. Raimondi, "A Rocket Launch for International Student Satellites," ARLISS, 2008.

[3] J. P. Zayas, "Rosetta CanSat Team," 7 March 2015. [Online]. Available: http://rosettacansat.blogspot.com.es/2015/03/titech-cansat-project-in-1999.html\.

[4] Cooking Hacks, "Geiger Counter - Radiation Sensor Board for Arduino and Raspberry Pi," [Online]. Available: https://www.cooking-hacks.com/documentation/tutorials/geiger-counter-radiation-sensor-board-arduino-raspberry-pi-tutorial/.

[5] ESA, "CanSat," 2016. [Online]. Available: http://www.cansat.eu\.

[6] Innovative Sensor Technology, "Platinum Temperature Sensors Datasheet," 2016.

[7] Philips Semiconductors, "Low power quad op amps Datasheet," 1995.

[8] Freescale Semiconductor, "200kPa On-Chip Temperature Compensated Silicon Pressure Sensors Datasheet," 2008.

[9] Maxim Integrated Products, "CMOS Monolithic Voltage Converter," 1996.

[10] Digi International Inc., "XBee-Pro 868 Datasheet," 2015.

[11] Arduino, "XBee Shield," 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoXbeeShield.

[12] Linx, "ANT-868-CW-RAH-xxx Data Sheet," 2015.

[13] ALSROBOT, "XBee-Explorer-v15," 2010.

[14] J. Berenguer, "Antenna 868MHz 17dB," 2008.

[15] Digi International Inc., "XCTU Next Generation Configuration Platform for XBee/RF Solutions," 2016. [Online]. Available: https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu.

[16] H. Chaouchi, The Internet of Things, London: Wiley-ISTE, 2010.

[17] O. Corporation, "The Java EE 6 Tutorial," 2013. [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html.

[18] B. Taha-Ahmed, "UMTS-HSDPA in High Altitude Platforms (HAPs) communications with finite transmitted power and unequal cell's load," 2009.

[19] Vodafone Group, "Vodafone R216," 2015. [Online]. Available: http://www.vodafone.com/content/index/what/devices/vodafone-r216.html.

[20] Espressif Systems, "ESP8266EX Datasheet," 2015.

[21] STMicroelectronics, "Low drop fixed and adjustable positive voltage regulators," 2005.

[22] Microsoft, "Web Server (IIS) Overview," 2016. [Online]. Available: https://technet.microsoft.com/en-us/library/hh831725(v=ws.11).aspx.

[23] M. Brun, Incident at Sakhalin: The True mission of KAL Flight 007, New York, London: Four Walls Eight Windows, 1995.

[24] J. Á. Rodriguez, "GPS SIgnal Plan," 2011. [Online]. Available: http://www.navipedia.net/index.php/GPS_Signal_Plan.

[25] Remains of the Day, "COCOM Limits," 5 August 2015. [Online]. Available: http://www.eugenewei.com/blog/2015/8/3/cocom-limits.

[26] U-blox, "NEO-6 u-blox 6 GPS Modules Datasheet," 2011.

[27] K. Betke, "The NMEA 0183 Protocol," 2000.

[28] Arduino, "SoftwareSerial Library," 2016. [Online]. Available: https://www.arduino.cc/en/Reference/SoftwareSerial.

[29] mikalhart, "A compact Arduino NMEA (GPS) parsing library," 2016. [Online]. Available: https://github.com/mikalhart/TinyGPS.

[30] Texas Instruments, "LM35 Precision Centigrade Temperature Sensors Datasheet," 2015.

[31] Vishay Siliconix, "Power MOSFET IRF510 Datasheet," 2015.

[32] Energizer Holdings, Inc., "Product Datasheet Energizer 522," 2016.

[33] Farnell, "Battery Clip - 9V Datasheet," 2007.

[34] Prometec, "¿Cuánto Consume Arduino?," 2016. [Online]. Available: http://www.prometec.net/consumos-arduino/.

[35] Arduino, "Arduino UNO Datasheet," 2016.

[36] NAROM, "The CanSat Book," 2013.

[37] Agencia Espacial Mexicana (AEM), "Implementación y Operación CanSat Kit AEM-UNAM/RUE," 2013.

[38] ACTIS, "Soluciones de aislamiento ACTIS," 2016. [Online]. Available: http://www.aislamiento-actis.com/gammes-produits-actis.php?p=2&l=2.

[39] Energizer Holdings, Inc., "Product Datasheet Energizer L522," 2016.

[40] Stratoflights, "Weather Balloon 1600," 2016. [Online]. Available: https://www.stratoflights.com/en/shop/wetterballon/.

[41] Gobierno de España, "Agencia Estatal Boletín Oficial del Estado," 2014. [Online]. Available: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2014-6856.

[42] J. Brohm, "The Mathematics of Flat Parachutes," 2004.

ANNEXES

Annex A. TECHNICAL DATASHEETS

Energizer Zinc Batteries

PRODUCT DATASHEET

Energizer

1-800-383-7323 USA/CAN
www.energizer.com

ENERGIZER 522

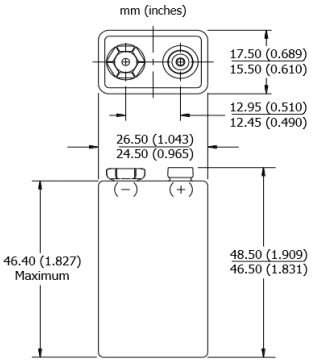
9V

Specifications

Classification:	Alkaline
Chemical System:	Zinc-Manganese Dioxide (Zn/MnO_2)
	No added mercury or cadmium
Designation:	ANSI-1604A, IEC-6LR61
Nominal Voltage:	9.0 volts
Operating Temp:	-18°C to 55°C (0°F to 130°F)
Typical Weight:	45.6 grams (1.6 oz.)
Typical Volume:	21.1 cubic centimeters (1.3 cubic inch)
Jacket:	Metal
Shelf Life:	5 years at 21°C
Terminal:	Miniature Snap

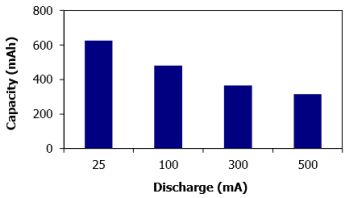
Industry Standard Dimensions

mm (inches)



Milliamp-Hours Capacity

Continuous discharge to 4.8 volts at 21°C

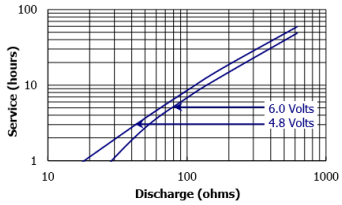


ENERGIZER 522

9V

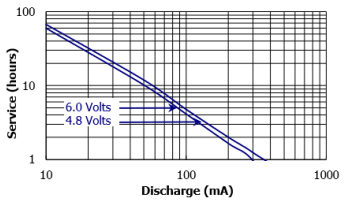
Constant Resistance Performance

Typical Characteristics (21°C)

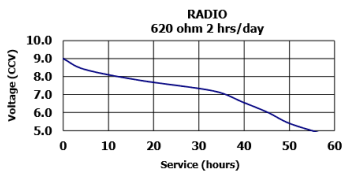


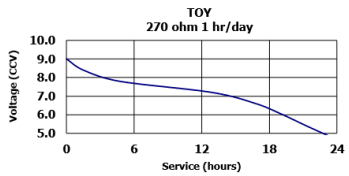
Constant Current Performance

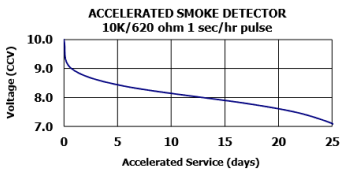
Typical Characteristics (21°C)



Industry Standard Tests (21°C)







Digi International XBee PRO 868

Platform	XBee-PRO® 868
Performance	
RF Data Rate	24 Kbps (limited to 10% duty cycle)
Indoor/Urban Range	Up to 1800 ft (550 m)
Outdoor/RF Line-of-Sight Range	Up to 25 miles (40 km) with dipole antenna (Italy only) up to 10 miles (16 km) with dipole antenna (13.7 dBm)
Transmit Power	1 mW (0 dBm) to 315 mW (+25 dBm)
Receiver Sensitivity (1% PER)	-112 dBm or 500 mW EIRP
Features	
Serial Data Interface	3.3V CMOS Serial UART
Configuration Method	API and AT commands
Frequency Band	868 MHz ISM
Interference Immunity	Multiple transmissions, acknowledgements
Serial Data Rate	1.2 Kbps to 230.4 Kbps (non-standard rates available)
ADC Inputs	6 (10-bit)
Digital I/O	13
Antenna Options	Wired Whip, U.FL connector, RPSMA connector
Networking & Security	
Encryption	128-bit AES
Reliable Packet Delivery	Retries/Acknowledgments
Addressing Options	Network ID, 64-bit address
Channels	Single channel
Power Requirements	
Supply Voltage	3.0 – 3.6VDC
Transmit Current	500 mA typical at 3.3V (800 mA max)
Receive Current	65 mA typical
Power-Down Current	55 uA typical @3.3V
Regulatory Approvals	
FCC (USA)	No
IC (Canada)	No
ETSI (Europe)	Yes (Italy 25 mW max)
C-TICK (Australia)	No
Telec (Japan)	No

Linx Technology 0.6 dB Antenna

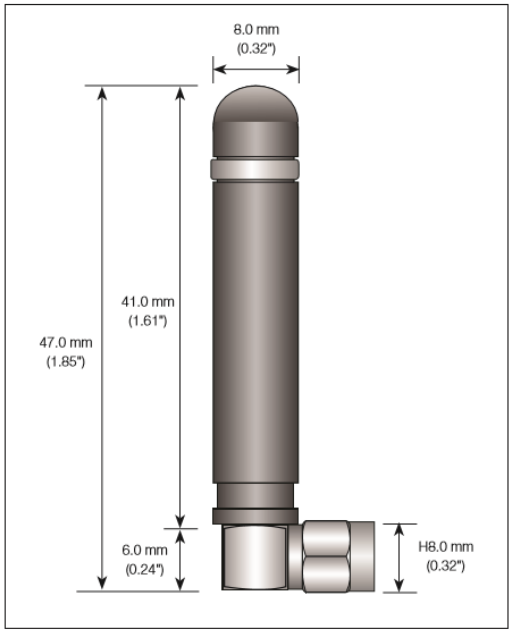
ANT-868-CW-RAH-xxx

Data Sheet

AntennaFactor by Linx

Product Description

The RAH Series utilizes a helical element to greatly reduce the physical length of the antenna housing. They are ideal for products requiring an ultra-compact, aesthetically pleasing antenna in a right angle form factor. Despite their tiny size, they are ruggedly constructed and able to withstand punishing environments just like our larger whips. The antennas attach via a standard SMA or Part 15 compliant RP-SMA connector.



Features

- Low cost
- Ultra-compact
- Right angle mount
- Excellent performance
- Omnidirectional pattern
- Fully weatherized
- Flexible main shaft
- Rugged & damage-resistant
- SMA or Part 15 compliant RP-SMA connector
- Use with plastic* or metal enclosures

* Requires proximity ground plane

Electrical Specifications

Center Frequency:	868MHz
Recom. Freq. Range:	833–903MHz
Wavelength:	¼-wave
VSWR:	≤ 1.9 typical at center
Peak Gain:	0.6dBi
Impedance:	50-ohms
Connector:	RP-SMA
Oper. Temp. Range:	–40° to +90°C

Electrical specifications and plots measured on 10.16 cm x 10.16 cm (4.00" x 4.00") reference ground plane

YAGI Antenna

Antena UHF Ref. Televés 1095 – DAT 45

2/4

CARACTERÍSTICAS TÉCNICAS

Referencias		**1095**
Elementos		45
Canal		21-69
Ganancia	dB	17
Relación D/A	dB	28
Longitud	mm	1020
Carga al viento (N)	785 N/m²	30
	1080 N/m²	43

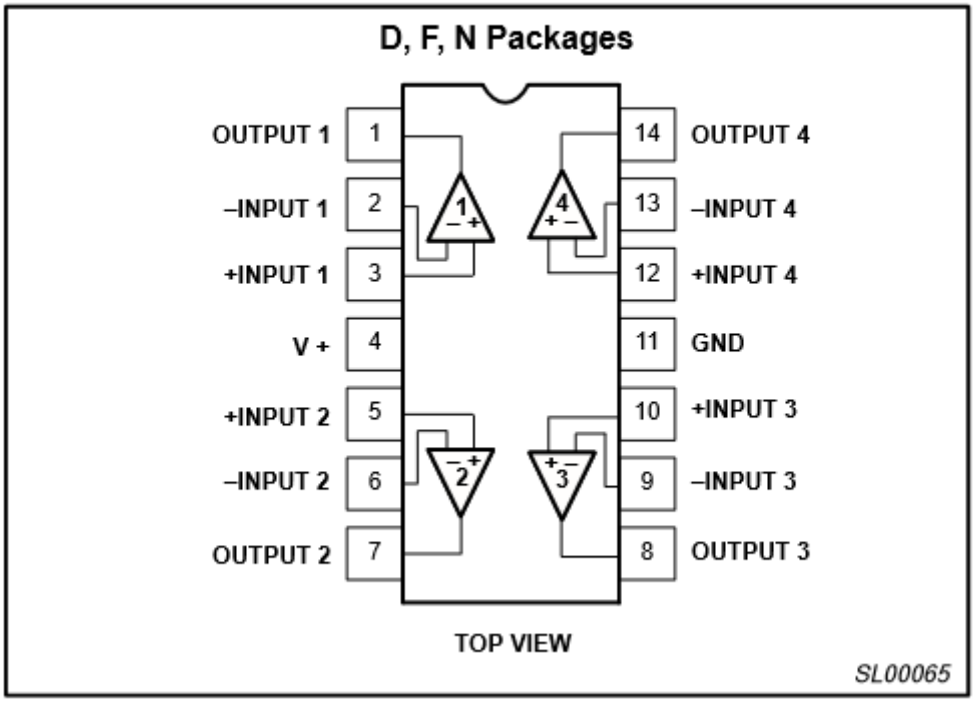
IST RTD Pt-1000 Temperature Sensor

7W 308

Dimensions, LxW:	3.0 x 0.8 mm	
Nominal Resistance at 0°C (ohm):	100/500/1000	
Self Heating (mK):	Water (v= 0 m/s) Air (v= 0 m/s)	ΔT_w = 6.7 at 0°C ΔT_a = 46 at 0°C
Response Time (s):	Water (v= 0.4 m/s)	$T_{0.5}$ = 0.08 $T_{0.63}$ = 0.10 $T_{0.9}$ = 0.25
	Air (v= 1 m/s)	$T_{0.5}$ = 1.2 $T_{0.63}$ = 1.5 $T_{0.9}$ = 3.5
Measuring Current (mA):	100 Ω: 1 500 Ω: 0.5 1000 Ω: 0.3	
Note:	Pure platinum wire, 0.15 mm diameter	

Phillips Semiconductor LM324N OPAMP

PIN CONFIGURATION



ABSOLUTE MAXIMUM RATINGS

SYMBOL	PARAMETER	RATING	UNIT
V_{CC}	Supply voltage	32 or ±16	V_{DC}
V_{IN}	Differential input voltage	32	V_{DC}
V_{IN}	Input voltage	-0.3 to +32	V_{DC}
P_D	Maximum power dissipation, T_A=25°C (still-air)[1]		
	N package	1420	mW
	F package	1190	mW
	D package	1040	mW
	Output short-circuit to GND one amplifier[2] V_{CC}<15V_{DC} and T_A=25°C	Continuous	
I_{IN}	Input current (V_{IN}<-0.3V)[3]	50	mA
T_A	Operating ambient temperature range		
	LM324/A	0 to +70	°C
	LM224	-25 to +85	°C
	SA534	-40 to +85	°C
	LM2902	-40 to +125	°C
	LM124	-55 to +125	°C
T_{STG}	Storage temperature range	-65 to +150	°C
T_{SOLD}	Lead soldering temperature (10sec max)	300	°C

Freescale MPX2200AP Absolute Pressure Sensor

Operating Characteristics

Table 1. Operating Characteristics (V_S = 10 V_{DC}, T_A = 25°C unless otherwise noted, P1 > P2)

Characteristic	Symbol	Min	Typ	Max	Units
Differential Pressure Range(1)	P_{OP}	0	—	200	kPa
Supply Voltage(2)	V_S	—	10	16	V_{DC}
Supply Current	I_O	—	6.0	—	mAdc
Full Scale Span(3)	V_{FSS}	38.5	40	41.5	mV
Offset(4)	V_{OFF}	-1.0	—	1.0	mV
Sensitivity	$\Delta V/\Delta P$	—	0.2	—	mV/kPa
Linearity MPX2200D Series MPX2200A Series	—	-0.25 -1.0	— —	0.25 1.0	%V_{FSS}
Pressure Hysteresis(0 to 200 kPa)	—	—	±0.1	—	%V_{FSS}
Temperature Hysteresis(- 40°C to +125°C)	—	—	±0.5	—	%V_{FSS}
Temperature Coefficient of Full Scale Span	TCV_{FSS}	-1.0	—	1.0	%V_{FSS}
Temperature Coefficient of Offset	TCV_{OFF}	-1.0	—	1.0	mV
Input Impedance	Z_{IN}	1300	—	2500	Ω
Output Impedance	Z_{OUT}	1400	—	3000	Ω
Response Time(5) (10% to 90%)	t_R	—	1.0	—	ms
Warm-Up Time(6)	—	—	20	—	ms
Offset Stability(7)	—	—	±0.5	—	%V_{FSS}

Maxim MAX660 Monolithic CMOS Voltage inverter

ABSOLUTE MAXIMUM RATINGS

Supply Voltage (V+ to GND, or GND to OUT) .....+6V
LV Input Voltage .....(OUT - 0.3V) to (V+ + 0.3V)
FC and OSC Input Voltages.....The least negative of (OUT - 0.3V) or (V+ - 6V) to (V+ + 0.3V)
OUT and V+ Continuous Output Current.....120mA
Output Short-Circuit Duration to GND (Note 1) 1sec
Continuous Power Dissipation (T_A = +70°C)
Plastic DIP (derate 9.09mW/°C above + 70°C) 727mW
SO (derate 5.88mW/°C above +70°C).....471mW
CERDIP (derate 8.00mW/°C above +70°C).....640mW

Operating Temperature Ranges
MAX660C_ _ .....0°C to +70°C
MAX660E_ _ .....-40°C to +85°C
MAX660MJA .....-55°C to +125°C
Storage Temperature Range..... -65°to +160°C
Lead Temperature (soldering, 10sec) ..... +300°C

Note 1: OUT may be shorted to GND for 1sec without damage, but shorting OUT to V+ may damage the device and should be avoided. Also, for temperatures above +85°C, OUT must not be shorted to GND or V+, even instantaneously, or device damage may result.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

(V+ = 5V, C1 = C2 = 150µF, test circuit of Figure 1, FC = open, $T_A = T_{MIN}$ to T_{MAX}, unless otherwise noted.) (Note 2)

PARAMETER			MIN	TYP	MAX	UNITS
Operating Supply Voltage	R_L = 1kΩ	Inverter, LV = open	3.0		5.5	V
		Inverter, LV = GND	1.5		5.5	
		Doubler, LV = OUT	2.5		5.5	
Supply Current	No load	FC = open, LV = open		0.12	0.5	mA
		FC = V+, LV = open		1	3	
Output Current	$T_A \le$ +85°C, OUT more negative than -4V		100			mA
	$T_A >$ +85°C, OUT more negative than -3.8V		100			
Output Resistance (Note 3)	I_L = 100mA	$T_A \le$ +85°C, C1 = C2 = 10µF, FC = V+ (Note 4)			15	Ω
		$T_A \le$ +85°C, C1 = C2 = 150µF		6.5	10.0	
		$T_A \le$ +85°C			12	
Oscillator Frequency	FC = open		5	10		kHz
	FC = V+		40	80		
OSC Input Current	FC = open			±1		µA
	FC = V+			±8		
Power Efficiency	R_L = 1kΩ connected between V+ and OUT		96	98		%
	R_L = 500Ω connected between OUT and GND		92	96		
	I_L = 100mA to GND			88		
Voltage-Conversion Efficiency	No load		99.00	99.96		%

Texas Instruments LM35 Temperature Sensor

6.7 Electrical Characteristics: LM35, LM35C, LM35D Limits

Unless otherwise noted, these specifications apply: −55°C ≤ T_J ≤ 150°C for the LM35 and LM35A; −40°C ≤ T_J ≤ 110°C for the LM35C and LM35CA; and 0°C ≤ T_J ≤ 100°C for the LM35D. V_S = 5 Vdc and I_{LOAD} = 50 µA, in the circuit of Full-Range Centigrade Temperature Sensor. These specifications also apply from 2°C to T_{MAX} in the circuit of Figure 14.

PARAMETER	TEST CONDITIONS	LM35			LM35C, LM35D			UNIT
		TYP	TESTED LIMIT(1)	DESIGN LIMIT(2)	TYP	TESTED LIMIT(1)	DESIGN LIMIT(2)	
Accuracy, LM35, LM35C(3)	T_A = 25°C	±0.4	±1		±0.4	±1		°C
	T_A = –10°C	±0.5			±0.5		±1.5	
	T_A = T_{MAX}	±0.8	±1.5		±0.8		±1.5	
	T_A = T_{MIN}	±0.8		±1.5	±0.8		±2	
Accuracy, LM35D(3)	T_A = 25°C				±0.6	±1.5		°C
	T_A = T_{MAX}				±0.9		±2	
	T_A = T_{MIN}				±0.9		±2	
Nonlinearity(4)	T_{MIN} ≤ T_A ≤ T_{MAX}, –40°C ≤ T_J ≤ 125°C	±0.3		±0.5	±0.2		±0.5	°C
Sensor gain (average slope)	T_{MIN} ≤ T_A ≤ T_{MAX}, –40°C ≤ T_J ≤ 125°C	10	9.8		10		9.8	mV/°C
		10	10.2		10		10.2	
Load regulation(5) 0 ≤ I_L ≤ 1 mA	T_A = 25°C	±0.4	±2		±0.4	±2		mV/mA
	T_{MIN} ≤ T_A ≤ T_{MAX}, –40°C ≤ T_J ≤ 125°C	±0.5		±5	±0.5		±5	
Line regulation(5)	T_A = 25°C	±0.01	±0.1		±0.01	±0.1		mV/V
	4 V ≤ V_S ≤ 30 V, –40°C ≤ T_J ≤ 125°C	±0.02		±0.2	±0.02		±0.2	
Quiescent current(6)	V_S = 5 V, 25°C	56	80		56	80		µA
	V_S = 5 V, –40°C ≤ T_J ≤ 125°C	105		158	91		138	
	V_S = 30 V, 25°C	56.2	82		56.2	82		
	V_S = 30 V, –40°C ≤ T_J ≤ 125°C	105.5		161	91.5		141	
Change of quiescent current(5)	4 V ≤ V_S ≤ 30 V, 25°C	0.2	2		0.2	2		µA
	4 V ≤ V_S ≤ 30 V, –40°C ≤ T_J ≤ 125°C	0.5		3	0.5		**3**	
Temperature coefficient of quiescent current	–40°C ≤ T_J ≤ 125°C	0.39		0.7	0.39		0.7	µA/°C
Minimum temperature for rate accuracy	In circuit of Figure 14, I_L = 0	1.5		2	1.5		2	°C
Long term stability	T_J = T_{MAX}, for 1000 hours	±0.08			±0.08			°C

Vishay IRF510 power MOSFET

SPECIFICATIONS (T_J = 25 °C, unless otherwise noted)						
PARAMETER	**SYMBOL**	**TEST CONDITIONS**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
Static						
Drain-Source Breakdown Voltage	V_{DS}	V_{GS} = 0 V, I_D = 250 µA	100	-	-	V
V_{DS} Temperature Coefficient	$\Delta V_{DS}/T_J$	Reference to 25 °C, I_D = 1 mA	-	0.12	-	V/°C
Gate-Source Threshold Voltage	$V_{GS(th)}$	V_{DS} = V_{GS}, I_D = 250 µA	2.0	-	4.0	V
Gate-Source Leakage	I_{GSS}	V_{GS} = ± 20 V	-	-	± 100	nA
Zero Gate Voltage Drain Current	I_{DSS}	V_{DS} = 100 V, V_{GS} = 0 V	-	-	25	µA
		V_{DS} = 80 V, V_{GS} = 0 V, T_J = 150 °C	-	-	250	
Drain-Source On-State Resistance	$R_{DS(on)}$	V_{GS} = 10 V, I_D =3.4 A [b]	-	-	0.54	Ω
Forward Transconductance	g_{fs}	V_{DS} = 50 V, I_D = 3.4 A [b]	1.3	-	-	S
Dynamic						
Input Capacitance	C_{iss}	V_{GS} = 0 V, V_{DS} = 25 V, f = 1.0 MHz, see fig. 5	-	180	-	pF
Output Capacitance	C_{oss}		-	81	-	
Reverse Transfer Capacitance	C_{rss}		-	15	-	
Total Gate Charge	Q_g	V_{GS} = 10 V, I_D = 5.6 A, V_{DS} = 80 V	-	-	8.3	nC
Gate-Source Charge	Q_{gs}	V_{DS} = 10 V, see fig. 6 and fig. 13 [b]	-	-	2.3	
Gate-Drain Charge	Q_{gd}		-	-	3.8	
Turn-On Delay Time	$t_{d(on)}$	V_{DD} = 50 V, I_D = 5.6 A, R_g = 24 Ω, R_D = 8.4 Ω, see fig. 10 [b]	-	6.9	-	ns
Rise Time	t_r		-	16	-	
Turn-Off Delay Time	$t_{d(off)}$		-	15	-	
Fall Time	t_f		-	9.4	-	
Internal Drain Inductance	L_D	Between lead, 6 mm (0.25") from package and center of die contact 	-	4.5	-	nH
Internal Source Inductance	L_S		-	7.5	-	
Drain-Source Body Diode Characteristics						
Continuous Source-Drain Diode Current	I_S	MOSFET symbol showing the integral reverse p - n junction diode 	-	-	5.6	A
Pulsed Diode Forward Current[a]	I_{SM}		-	-	20	
Body Diode Voltage	V_{SD}	T_J = 25 °C, I_S = 5.6 A, V_{GS} = 0 V [b]	-	-	2.5	V
Body Diode Reverse Recovery Time	t_{rr}	T_J = 25 °C, I_F = 5.6 A, dI/dt = 100 A/µs [b]	-	100	200	ns
Body Diode Reverse Recovery Charge	Q_{rr}		-	0.44	0.88	µC
Forward Turn-On Time	t_{on}	Intrinsic turn-on time is negligible (turn-on is dominated by L_S and L_D)				

U-blox GPS module

1.3 GPS performance

Parameter	Specification			
Receiver type	50 Channels GPS L1 frequency, C/A Code SBAS: WAAS, EGNOS, MSAS			
Time-To-First-Fix[1]		NEO-6G/Q/T	NEO-6M/V	NEO-6P
	Cold Start[2]	26 s	27 s	32 s
	Warm Start[2]	26 s	27 s	32 s
	Hot Start[2]	1 s	1 s	1 s
	Aided Starts[3]	1 s	<3 s	<3 s
Sensitivity[4]		NEO-6G/Q/T	NEO-6M/V	NEO-6P
	Tracking & Navigation	-162 dBm	-161 dBm	-160 dBm
	Reacquisition[5]	-160 dBm	-160 dBm	-160 dBm
	Cold Start (without aiding)	-148 dBm	-147 dBm	-146 dBm
	Hot Start	-157 dBm	-156 dBm	-155 dBm
Maximum Navigation update rate		NEO-6G/Q/M/T	NEO-6P/V	
		5Hz	1 Hz	
Horizontal position accuracy[6]	GPS	2.5 m		
	SBAS	2.0 m		
	SBAS + PPP[7]	< 1 m (2D, R50)[8]		
	SBAS + PPP[7]	< 2 m (3D, R50)[8]		
Configurable Timepulse frequency range		NEO-6G/Q/M/P/V	NEO-6T	
		0.25 Hz to 1 kHz	0.25 Hz to 10 MHz	
Accuracy for Timepulse signal	RMS	30 ns		
	99%	<60 ns		
	Granularity	21 ns		
	Compensated[9]	15 ns		
Velocity accuracy[6]		0.1m/s		
Heading accuracy[6]		0.5 degrees		
Operational Limits	Dynamics	≤ 4 g		
	Altitude[10]	50,000 m		
	Velocity[10]	500 m/s		

ESP8266 Wi-Fi module

Categories	Items	Values
WiFi Paramters	Certificates	FCC/CE/TELEC/SRRC
	WiFi Protocles	802.11 b/g/n
	Frequency Range	2.4G-2.5G (2400M-2483.5M)
	Tx Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: -91 dbm (11 Mbps)
		802.11 g: -75 dbm (54 Mbps)
		802.11 n: -72 dbm (MCS7)
	Types of Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware Paramaters	Peripheral Bus	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/PWM
	Operating Voltage	3.0~3.6V
	Operating Current	Average value: 80mA
	Operating Temperature Range	-40°~125°
	Ambient Temperature Range	Normal temperature
	Package Size	5x5mm
	External Interface	N/A
Software Parameters	WiFi mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Ssoftware Development	Supports Cloud Server Development / SDK for custom firmware development
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP

3.3V voltage regulator

Table 9: Electrical Characteristics Of LD1117#33 (refer to the test circuits, T_J = 0 to 125°C, C_O = 10 µF unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_O	Output Voltage	V_{in} = 5.3 V I_O = 10 mA T_J = 25°C	3.267	3.3	3.333	V
V_O	Output Voltage	I_O = 0 to 800 mA V_{in} = 4.75 to 10 V	3.235		3.365	V
ΔV_O	Line Regulation	V_{in} = 4.75 to 15 V I_O = 0 mA		1	6	mV
ΔV_O	Load Regulation	V_{in} = 4.75 V I_O = 0 to 800 mA		1	10	mV
ΔV_O	Temperature Stability			0.5		%
ΔV_O	Long Term Stability	1000 hrs, T_J = 125°C		0.3		%
V_{in}	Operating Input Voltage	I_O = 100 mA			15	V
I_d	Quiescent Current	$V_{in} \leq$ 15 V		5	10	mA
I_O	Output Current	V_{in} = 8.3 V T_J = 25°C	800	950	1300	mA
eN	Output Noise Voltage	B =10Hz to 10KHz T_J = 25°C		100		µV
SVR	Supply Voltage Rejection	I_O = 40 mA f = 120Hz T_J = 25°C V_{in} = 6.3 V V_{ripple} = 1 V_{PP}	60	75		dB
V_d	Dropout Voltage	I_O = 100 mA		1	1.1	V
		I_O = 500 mA		1.05	1.15	
		I_O = 800 mA		1.10	1.2	
	Thermal Regulation	T_a = 25°C 30ms Pulse		0.01	0.1	%/W

Arduino UNO board

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V

Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

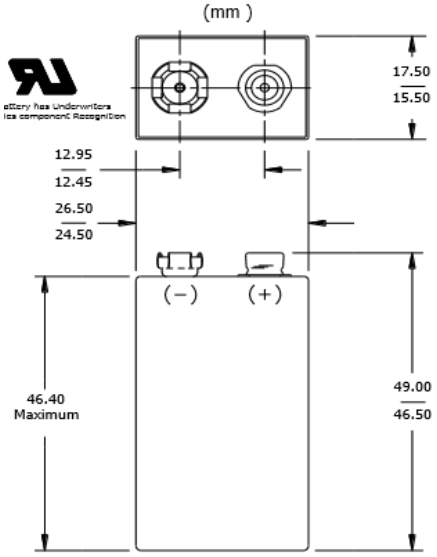
Energizer Lithium 9V Battery

ENERGIZER L522

Lithium

9V

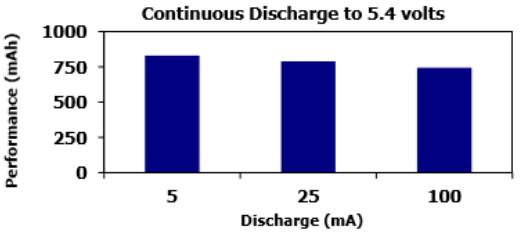
Industry Standard Dimensions

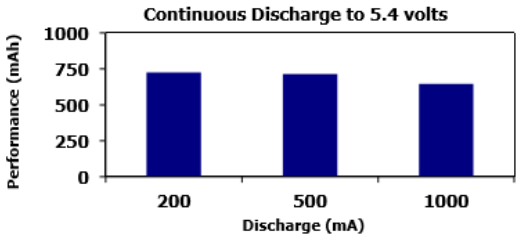


Specifications

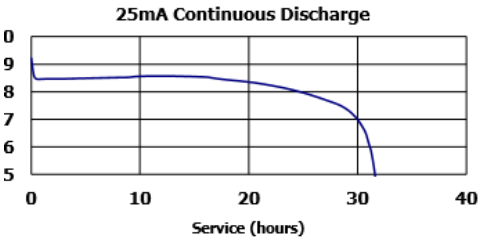
Classification:	Lithium 9V
Chemical System:	Lithium-Manganese Dioxide (Li/MnO_2)
Designation:	ANSI-1604LC
Nominal Voltage:	9.0 volts
Operating Temp:	-40°C to 60°C
Storage Temp:	-40°C to 60°C
Max Discharge :	1000 mA continuous
Safety Features:	Positive Temperature Coefficient Switch (PTC) Burst Proof Venting Holes
Typical Weight:	33.9 grams
Typical Volume:	21.4 cubic centimeters
Jacket:	Plastic Label
Terminal:	Miniature Snap
Shelf Life:	10 Years
Typical Li Content:	1.35 grams
UL/UN Listed:	MH12454 / 38.3

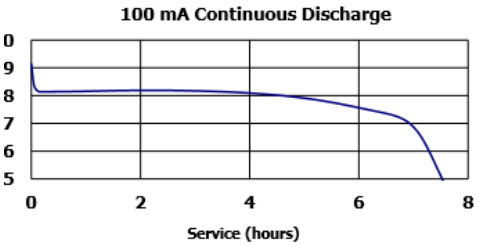
Milliamp-Hours Performance (21°C)

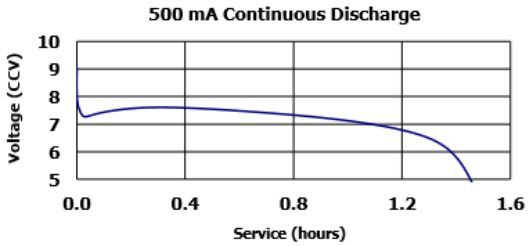




ypical Discharge Performance (21°C)







Annex B. ARDUINO AND SERVER CODE

Arduino code

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>

// Initialize variables and configure SoftwareSerial library
TinyGPS gps;
SoftwareSerial GNSS(5, 4);
SoftwareSerial ESP(3, 2);
int req = 0;
String dates, times, txtimestamp;
float flat, flon, falt;

void setup()
{
  Serial.begin(9600);
  GNSS.begin(9600);
  ESP.begin(9600);
  ESP.println("AT+RST");
  analogReference(INTERNAL);
  Serial.println("Starting up systems...");
  pinMode(10, OUTPUT);
  delay(10000);


}

void loop()
{
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;
  GNSS.listen();

  // For one second we parse GPS data and report some key values
  for (unsigned long start = millis(); millis() - start < 1000;)
  {
    while (GNSS.available())
    {
      char c = GNSS.read();
      if (gps.encode(c)) // Did a new valid sentence come in?
        newData = true;
    }
  }

// Dummy values for the ESP request when there is no GNSS data
  dates ="311299";
  times = "00000000";
  flat=89.56;
  flon=44.56;
  falt=-100.56;

  if (newData)// Only print data into serial monitor if valid NMEA sentences are recieved
  {
```

```
  unsigned long age;
  unsigned long fix_age, time, date;
  gps.f_get_position(&flat, &flon, &age);
  gps.get_datetime(&date, &time, &fix_age);
  falt = gps.f_altitude();
  Serial.print("LT=");//Latitude
  Serial.println(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
  delay(100);
  Serial.print("LN="); //Longitude
  Serial.println(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
  delay(100);
  Serial.print("AL="); //Altitude
  Serial.println(falt == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : falt, 6);
  delay(100);
  Serial.print("DT="); //Date
  Serial.println(date == TinyGPS::GPS_INVALID_F_ANGLE ? 0 : date);
  delay(100);
  Serial.print("TM="); //Time
  Serial.println(time == TinyGPS::GPS_INVALID_F_ANGLE ? 0 : time);
  delay(100);
  if (String(date).length() == 5) //Format correctly date and time for the server code.
 dates = "0"+String(date);

 else
 dates = String(date);

 if (String(time).length() == 7)
 times = "0"+String(time);

 else
 times = String(time);

}
txtimestamp = dates+times;
int Setemp, Sitemp, Sepres;
float fetemp, fitemp, fepres;
unsigned long scount = 0;
Setemp = 0;
Sitemp = 0;
Sepres = 0;


//DCA 17/09 in order to reduce noise, we make 15 samples for each measure, in two seconds.
 while (scount < 15)
 {
  Setemp += analogRead(A3);
  delay(200);
  scount++;
 }
 scount = 0;
 Setemp = Setemp/15;

  while (scount < 10)
 {
  Sitemp += analogRead(A1);
  delay(200);
  scount++;
 }
 scount = 0;
 Sitemp = Sitemp/10;
```

```
  while (scount < 15)
 {
  Sepres += analogRead(A2);
  delay(200);
  scount++;
 }
 scount = 0;
 Sepres = Sepres/15;
 fetemp = (((Setemp * (1.1 / 1023.0)) * 1000)/11)-65; //External Temperature
 fitemp = (Sitemp * (1.1 / 1023.0)) * 100;          //Internal Temperature
 fepres = ((Sepres * (1.1 / 1023.0)) * 1000)/9.546;   //External Pressure

 Serial.print("ET=");
 Serial.println(fetemp);
 delay(100);
 Serial.print("IT=");
 Serial.println(fitemp);
 delay(100);
 Serial.print("EP=");
 Serial.println(fepres);

 delay(1000);

//Heating system activates if below 2 degrees and deactivates above 3 degrees
 bool heaton;
 String heatons;
 if (fitemp <= 2)
 {
 heaton = true;
 heatons = "true";
 }
 if (fitemp >= 3)
 {
 heaton = false;
 heatons="false";
 }

 if (heaton)
 {
 digitalWrite(10,HIGH);
 Serial.println("Heat ON");
 }

 if(!heaton)
 {
 digitalWrite(10,LOW);
 Serial.println("Heat OFF");
 }

 ESP.listen(); //We prepare the HTTP petition for the target server.

 delay(1000);
 ESP.println("AT+CIPSTART=\"TCP\",\"83.39.119.76\",80"); //Open Connection socket
 delay(2000);
 ESP.println("AT+CIPSEND=158"); //Size of the data sent.
 delay(5000);
 ESP.println("GET
/RestMissionControl/MissionControl.svc/data/"+String(req)+"/"+txtimestamp+"/"+String(fetemp,2
```

```
)+"/"+String(fitemp,2)+"/"+String(fepres,2)+"/"+String(flat,6)+"/"+String(flon,6)+"/"+String(falt,2)+"
/"+heatons+" HTTP/1.1"); //HTTP GET petition//
  ESP.println("HOST: 83.39.119.76");
  ESP.println();
  ESP.println();
  delay(50);
  ESP.println();
  ESP.println(); //Additional line breaks in order to force sending the request.
  delay(50);
  ESP.println();
  ESP.println();
  delay(50);
  ESP.println();
  ESP.println();
  delay(50);
  ESP.println();
  delay(50);
  ESP.println();
  Serial.println("SQ"+String(req));
  delay(500);
  req+=1;
  ESP.println("AT+CIPCLOSE"); //Close the socket
  delay(10000);

}
```

Server code (WCF RESTFul service)

MissionControl.svc.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data.SqlClient;
using System.Globalization;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace RestMissionControl
{
   // NOTA: puede usar el comando "Rename" del menú "Refactorizar" para
cambiar el nombre de clase "MissionControl" en el código, en svc y en el archivo
de configuración a la vez.
   // NOTA: para iniciar el Cliente de prueba WCF para probar este servicio,
seleccione MissionControl.svc o MissionControl.svc.cs en el Explorador de
soluciones e inicie la depuración.
   public class MissionControl : IMissionControl
   {
      public string ConString =
ConfigurationManager.ConnectionStrings["ConString"].ConnectionString;
      public string missiondata(string Txseq, string TxTimestamp, string ExtTemp,
string IntTemp, string ExtPres, string Lat, string Lon, string Alt, string HeatOn)
```

```
    {
        Boolean HeatOnb = Convert.ToBoolean(HeatOn);
        long Txseqi = Convert.ToInt16(Txseq);
        string RxTimestamp = DateTime.UtcNow.ToString("yyyy-dd-MM
HH:mm:ss");
        string cc = TxTimestamp.Substring(TxTimestamp.Length - 2, 2);
        string ss = TxTimestamp.Substring(TxTimestamp.Length - 4, 2);
        string mm = TxTimestamp.Substring(TxTimestamp.Length - 6, 2);
        string hh = TxTimestamp.Substring(TxTimestamp.Length - 8, 2);
        string yy = TxTimestamp.Substring(TxTimestamp.Length - 10, 2);
        string MM = TxTimestamp.Substring(TxTimestamp.Length - 12, 2);
        string dd = TxTimestamp.Substring(TxTimestamp.Length - 14, 2);

        string TxTimestampdate = "20" + yy + "-" +dd + "-" + MM + " " + hh + ":" +
mm + ":" + ss;

        string cmd = string.Format("INSERT INTO dbo.MissionData
([RxTimestamp], [ExtTemp], [IntTemp], [ExtPres], [Lat], [Lon], [Alt], [HeatOn],
[TxSeq], [TxTimestamp]) Values (CAST('{0}' AS
DATETIME),'{1}','{2}','{3}','{4}','{5}','{6}',CAST('{7}' AS BIT), {8} , CAST('{9}' AS
DATETIME))", RxTimestamp, ExtTemp, IntTemp, ExtPres, Lat, Lon, Alt,
HeatOnb, Txseqi, TxTimestampdate);


        using (SqlConnection userconn = new SqlConnection(ConString))
        {
            try
            {

                userconn.Open();


                SqlCommand insertData = new SqlCommand(cmd, userconn);
                //SqlCommand insertData = new
SqlCommand(string.Format("INSERT INTO dbo.MissionData ([RxTimestamp],
[TxTimestamp], [ExtTemp], [IntTemp], [ExtPres], [Lat], [Lon], [Alt], [HeatOn],
[TxSeq]) Values (CAST('{0}' AS NVARCHAR), CAST('{1}' AS NVARCHAR),
CAST('{2}' AS FLOAT), CAST('{3}' AS FLOAT), CAST('{4}' AS FLOAT),
CAST('{5}' AS FLOAT), CAST('{6}' AS FLOAT), CAST('{7}' AS FLOAT),
CAST('{8}' AS BIT), CAST('{9}' AS INT))", RxTimestamp, TxTimestamp,
ExtTempd, IntTempd, ExtPresd, Latd, Lond, Altd, HeatOnb, Txseqi), userconn);

                if (insertData.ExecuteNonQuery() == 0)
                    throw new Exception("Mission Data Cannot be inserted");

            }

            catch (Exception ex)
            {
```

```
                throw new Exception(string.Format(@"Cannot insert mision data, exception: \n {0}", ex.Message));
            }

            return String.Format("Correct {0}", Txseq);
        }
    }
  }
}
```

IMissionControl.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace RestMissionControl
{
  // NOTA: puede usar el comando "Rename" del menú "Refactorizar" para cambiar el nombre de interfaz "IMissionControl" en el código y en el archivo de configuración a la vez.
  [ServiceContract]
  public interface IMissionControl
  {
    [OperationContract]
    [WebInvoke(Method = "GET",
      ResponseFormat = WebMessageFormat.Json,
      BodyStyle = WebMessageBodyStyle.Wrapped,
      UriTemplate = "data/{Txseq}/{TxTimestamp}/{ExtTemp}/{IntTemp}/{ExtPres}/{Lat}/{Lon}/{Alt}/{HeatOn}")]
    string missiondata(string Txseq, string TxTimestamp, string ExtTemp, string IntTemp, string ExtPres, string Lat, string Lon, string Alt, string HeatOn);

  }
}
```

Web.config

```
<?xml version="1.0"?>
<configuration>

  <system.web>
    <compilation debug="true" targetFramework="4.0" />
```

```
  </system.web>

  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source = |SQL/CE|"
/>
    <add name="ConString" connectionString="server=localhost;
                                Integrated Security=SSPI;
                                database=MissionControl;
                                MultipleActiveResultSets=True;
                                Connection Timeout=120" />
  </connectionStrings>

  <system.serviceModel>

    <services>
      <service                              name="RestMissionControl.MissionControl"
behaviorConfiguration="ServiceBehavior">
        <endpoint                                                  binding="webHttpBinding"
contract="RestMissionControl.IMissionControl"
                behaviorConfiguration="webHttp"/>
      </service>
    </services>

    <behaviors>
      <serviceBehaviors>

        <behavior name="ServiceBehavior" >
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>


        <behavior>
          <!-- To avoid disclosing metadata information, set the value below to false
and remove the metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="true"/>
          <!-- To receive exception details in faults for debugging purposes, set the
value below to true.  Set to false before deployment to avoid disclosing exception
information -->
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>

      <endpointBehaviors>
        <behavior name="webHttp">
          <webHttp/>
        </behavior>
      </endpointBehaviors>

    </behaviors>
```

```
  </system.serviceModel>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true"/>
  </system.webServer>

</configuration>
```

Annex C. FREEZER TEST RESULTS

One iteration every 30 seconds.

Starting up systems...
E.TEMP=35.00 I.TEMP=30.65 E.PRES=101.60
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/400/31129900000000/35.00/30.65/101.60/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=18.48 I.TEMP=30.75 E.PRES=104.42
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/401/31129900000000/18.48/30.75/104.42/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=31.48 I.TEMP=30.00 E.PRES=102.84
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/402/31129900000000/31.48/30.00/102.84/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=31.87 I.TEMP=31.40 E.PRES=101.15
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/403/31129900000000/31.87/31.40/101.15/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=32.36 I.TEMP=31.51 E.PRES=102.50
GET
/RestMissionControl/MissionControl.svc/data/404/31129900000000/32.36/31.51/102.50/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=32.56 I.TEMP=30.75 E.PRES=101.38
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/405/31129900000000/32.56/30.75/101.38/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=32.65 I.TEMP=31.08 E.PRES=102.17
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/406/31129900000000/32.65/31.08/102.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=28.65 I.TEMP=31.29 E.PRES=102.05
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/407/31129900000000/28.65/31.2
9/102.05/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.34 I.TEMP=31.29 E.PRES=101.04
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/408/31129900000000/33.34/31.2
9/101.04/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.22 I.TEMP=31.40 E.PRES=102.50
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/409/31129900000000/34.22/31.4
0/102.50/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.41 I.TEMP=32.15 E.PRES=102.39
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/410/31129900000000/34.41/32.1
5/102.39/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.71 I.TEMP=32.58 E.PRES=110.16
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/411/31129900000000/34.71/32.5
8/110.16/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.22 I.TEMP=32.04 E.PRES=102.50
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/412/31129900000000/34.22/32.0
4/102.50/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.61 I.TEMP=32.15 E.PRES=102.17
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/413/31129900000000/34.61/32.1
5/102.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.90 I.TEMP=32.69 E.PRES=102.05
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/414/31129900000000/34.90/32.6
9/102.05/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=32.04 E.PRES=101.83
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/415/31129900000000/35.00/32.0
4/101.83/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=33.01 E.PRES=101.60
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/416/31129900000000/35.00/33.0
1/101.60/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=32.90 E.PRES=101.38
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/417/31129900000000/35.00/32.9
0/101.38/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=33.23 E.PRES=101.60
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/418/31129900000000/35.00/33.2
3/101.60/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=33.23 E.PRES=102.05
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/419/31129900000000/35.00/33.2
3/102.05/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.41 I.TEMP=33.98 E.PRES=102.05
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/420/31129900000000/34.41/33.9
8/102.05/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.73 I.TEMP=33.66 E.PRES=100.14
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/421/31129900000000/33.73/33.6
6/100.14/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.04 I.TEMP=34.19 E.PRES=100.14
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/422/31129900000000/33.04/34.1
9/100.14/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.04 I.TEMP=34.09 E.PRES=101.94
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/423/31129900000000/33.04/34.0
9/101.94/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=31.87 I.TEMP=34.30 E.PRES=101.94
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/424/31129900000000/31.87/34.3
0/101.94/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=31.87 I.TEMP=34.62 E.PRES=102.17
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/425/31129900000000/31.87/34.6
2/102.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=31.58 I.TEMP=34.62 E.PRES=102.17
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/426/31129900000000/31.58/34.6
2/102.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=30.89 I.TEMP=34.41 E.PRES=102.28
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/427/31129900000000/30.89/34.4
1/102.28/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=30.11 I.TEMP=34.73 E.PRES=101.49
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/428/31129900000000/30.11/34.7
3/101.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=30.01 I.TEMP=34.30 E.PRES=102.84
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/429/31129900000000/30.01/34.3
0/102.84/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=29.72 I.TEMP=33.98 E.PRES=100.93
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/430/31129900000000/29.72/33.9
8/100.93/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=29.43 I.TEMP=34.19 E.PRES=101.60
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/431/31129900000000/29.43/34.19/101.60/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=28.94 I.TEMP=34.19 E.PRES=101.04
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/432/31129900000000/28.94/34.19/101.04/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=28.55 I.TEMP=34.73 E.PRES=101.49
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/433/31129900000000/28.55/34.73/101.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=28.35 I.TEMP=33.98 E.PRES=102.73
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/434/31129900000000/28.35/33.98/102.73/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=28.16 I.TEMP=33.87 E.PRES=101.83
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/435/31129900000000/28.16/33.87/101.83/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=26.89 I.TEMP=34.41 E.PRES=102.39
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/436/31129900000000/26.89/34.41/102.39/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=26.79 I.TEMP=33.44 E.PRES=102.05
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/437/31129900000000/26.79/33.44/102.05/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=27.47 I.TEMP=34.30 E.PRES=100.93
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/438/31129900000000/27.47/34.30/100.93/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=27.38 I.TEMP=34.30 E.PRES=102.95
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/439/31129900000000/27.38/34.30/102.95/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=26.50 I.TEMP=33.23 E.PRES=102.28
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/440/31129900000000/26.50/33.23/102.28/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=32.07 I.TEMP=33.01 E.PRES=102.05
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/441/31129900000000/32.07/33.01/102.05/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=31.38 I.TEMP=32.80 E.PRES=101.71
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/442/31129900000000/31.38/32.80/101.71/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=32.37 E.PRES=101.49
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/443/31129900000000/35.00/32.37/101.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.53 I.TEMP=32.90 E.PRES=101.26
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/444/31129900000000/33.53/32.90/101.26/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.51 I.TEMP=32.37 E.PRES=101.94
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/445/31129900000000/34.51/32.37/101.94/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.44 I.TEMP=31.72 E.PRES=100.81
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/446/31129900000000/33.44/31.72/100.81/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.41 I.TEMP=32.04 E.PRES=101.04
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/447/31129900000000/34.41/32.0
4/101.04/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.61 I.TEMP=31.94 E.PRES=101.71
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/448/31129900000000/34.61/31.9
4/101.71/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.34 I.TEMP=31.29 E.PRES=102.17
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/449/31129900000000/33.34/31.2
9/102.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=32.75 I.TEMP=31.40 E.PRES=103.29
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/450/31129900000000/32.75/31.4
0/103.29/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.61 I.TEMP=30.32 E.PRES=101.26
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/451/31129900000000/34.61/30.3
2/101.26/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=29.57 E.PRES=100.70
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/452/31129900000000/35.00/29.5
7/100.70/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=29.57 E.PRES=102.62
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/453/31129900000000/35.00/29.5
7/102.62/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.90 I.TEMP=29.89 E.PRES=100.93
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/454/31129900000000/34.90/29.8
9/100.93/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.80 I.TEMP=29.89 E.PRES=101.49
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/455/31129900000000/34.80/29.8
9/101.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=31.29 I.TEMP=28.92 E.PRES=101.94
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/456/31129900000000/31.29/28.9
2/101.94/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=32.95 I.TEMP=29.14 E.PRES=102.84
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/457/31129900000000/32.95/29.1
4/102.84/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.02 I.TEMP=28.39 E.PRES=102.17
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/458/31129900000000/34.02/28.3
9/102.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.32 I.TEMP=28.49 E.PRES=102.28
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/459/31129900000000/34.32/28.4
9/102.28/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=34.80 I.TEMP=27.96 E.PRES=101.26
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/460/31129900000000/34.80/27.9
6/101.26/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.92 I.TEMP=28.06 E.PRES=102.17
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/461/31129900000000/33.92/28.0
6/102.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=33.92 I.TEMP=27.10 E.PRES=102.17
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/462/31129900000000/33.92/27.1
0/102.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=27.53 E.PRES=100.25
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/463/31129900000000/35.00/27.53/100.25/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=26.88 E.PRES=101.94
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/464/31129900000000/35.00/26.88/101.94/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=26.50 I.TEMP=26.45 E.PRES=101.49
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/465/31129900000000/26.50/26.45/101.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=26.13 E.PRES=102.73
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/466/31129900000000/35.00/26.13/102.73/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=25.81 E.PRES=100.70
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/467/31129900000000/35.00/25.81/100.70/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=25.27 E.PRES=102.05
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/468/31129900000000/35.00/25.27/102.05/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=35.00 I.TEMP=26.24 E.PRES=99.69
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/469/31129900000000/35.00/26.24/99.69/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=16.04 I.TEMP=25.38 E.PRES=101.26
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/470/31129900000000/16.04/25.38/101.26/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=15.65 I.TEMP=25.16 E.PRES=102.62
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/471/31129900000000/15.65/25.16/102.62/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=15.55 I.TEMP=24.95 E.PRES=101.83
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/472/31129900000000/15.55/24.95/101.83/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=29.53 I.TEMP=24.62 E.PRES=102.62
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/473/31129900000000/29.53/24.62/102.62/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=28.35 I.TEMP=24.52 E.PRES=102.05
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/474/31129900000000/28.35/24.52/102.05/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=25.13 I.TEMP=24.19 E.PRES=100.03
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/475/31129900000000/25.13/24.19/100.03/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=26.50 I.TEMP=24.09 E.PRES=101.26
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/476/31129900000000/26.50/24.09/101.26/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=32.95 I.TEMP=23.76 E.PRES=101.26
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/477/31129900000000/32.95/23.76/101.26/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=22.68 I.TEMP=23.55 E.PRES=101.38
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/478/31129900000000/22.68/23.55/101.38/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=22.59 I.TEMP=23.66 E.PRES=101.49
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/479/31129900000000/22.59/23.6
6/101.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=23.07 I.TEMP=23.23 E.PRES=100.70
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/480/31129900000000/23.07/23.2
3/100.70/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=23.56 I.TEMP=23.23 E.PRES=99.80
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/481/31129900000000/23.56/23.2
3/99.80/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=25.22 I.TEMP=22.47 E.PRES=100.81
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/482/31129900000000/25.22/22.4
7/100.81/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=22.19 I.TEMP=22.47 E.PRES=101.71
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/483/31129900000000/22.19/22.4
7/101.71/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=20.34 I.TEMP=22.26 E.PRES=100.25
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/484/31129900000000/20.34/22.2
6/100.25/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=22.19 I.TEMP=22.15 E.PRES=100.59
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/485/31129900000000/22.19/22.1
5/100.59/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=22.98 I.TEMP=21.61 E.PRES=101.49
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/486/31129900000000/22.98/21.6
1/101.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=23.76 I.TEMP=20.97 E.PRES=101.83
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/487/31129900000000/23.76/20.97/101.83/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=26.40 I.TEMP=21.29 E.PRES=98.22
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/488/31129900000000/26.40/21.29/98.22/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=24.05 I.TEMP=21.61 E.PRES=98.90
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/489/31129900000000/24.05/21.61/98.90/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=18.68 I.TEMP=21.08 E.PRES=102.62
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/490/31129900000000/18.68/21.08/102.62/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=16.62 I.TEMP=21.18 E.PRES=101.26
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/491/31129900000000/16.62/21.18/101.26/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=21.80 I.TEMP=20.75 E.PRES=100.70
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/492/31129900000000/21.80/20.75/100.70/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=13.49 I.TEMP=20.43 E.PRES=102.50
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/493/31129900000000/13.49/20.43/102.50/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=13.69 I.TEMP=20.86 E.PRES=99.91
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/494/31129900000000/13.69/20.86/99.91/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=12.52 I.TEMP=19.57 E.PRES=100.14
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/495/31129900000000/12.52/19.57/100.14/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=11.93 I.TEMP=19.46 E.PRES=101.71
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/496/31129900000000/11.93/19.46/101.71/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=12.71 I.TEMP=19.89 E.PRES=98.56
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/497/31129900000000/12.71/19.89/98.56/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=11.93 I.TEMP=20.00 E.PRES=100.14
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/498/31129900000000/11.93/20.00/100.14/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=13.69 I.TEMP=18.92 E.PRES=100.93
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/499/31129900000000/13.69/18.92/100.93/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=13.79 I.TEMP=19.35 E.PRES=100.81
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/500/31129900000000/13.79/19.35/100.81/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=13.30 I.TEMP=18.71 E.PRES=99.80
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/501/31129900000000/13.30/18.71/99.80/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=22.78 I.TEMP=18.49 E.PRES=101.49
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/502/31129900000000/22.78/18.49/101.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=13.59 I.TEMP=17.85 E.PRES=101.38
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/503/31129900000000/13.59/17.85/101.38/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=18.58 I.TEMP=18.06 E.PRES=101.15
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/504/31129900000000/18.58/18.06/101.15/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=7.04 I.TEMP=18.06 E.PRES=101.38
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/505/31129900000000/7.04/18.06/101.38/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=6.26 I.TEMP=18.06 E.PRES=100.93
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/506/31129900000000/6.26/18.06/100.93/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=6.95 I.TEMP=18.17 E.PRES=100.93
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/507/31129900000000/6.95/18.17/100.93/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=7.04 I.TEMP=17.63 E.PRES=101.15
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/508/31129900000000/7.04/17.63/101.15/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=6.46 I.TEMP=17.53 E.PRES=100.48
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/509/31129900000000/6.46/17.53/100.48/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=6.36 I.TEMP=17.63 E.PRES=101.38
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/510/31129900000000/6.36/17.63/101.38/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=6.07 I.TEMP=17.63 E.PRES=98.11
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/511/31129900000000/6.07/17.63/98.11/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=5.09 I.TEMP=16.77 E.PRES=99.12
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/512/31129900000000/5.09/16.77/99.12/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=5.67 I.TEMP=16.67 E.PRES=100.93
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/513/31129900000000/5.67/16.67/100.93/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=6.07 I.TEMP=16.67 E.PRES=99.46
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/514/31129900000000/6.07/16.67/99.46/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=5.38 I.TEMP=16.56 E.PRES=100.59
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/515/31129900000000/5.38/16.56/100.59/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=5.19 I.TEMP=16.34 E.PRES=100.81
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/516/31129900000000/5.19/16.34/100.81/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=4.11 I.TEMP=16.13 E.PRES=99.57
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/517/31129900000000/4.11/16.13/99.57/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=4.79 I.TEMP=16.34 E.PRES=101.38
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/518/31129900000000/4.79/16.34/101.38/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=4.89 I.TEMP=15.91 E.PRES=99.01
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/519/31129900000000/4.89/15.91/99.01/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=4.50 I.TEMP=16.13 E.PRES=100.48
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/520/31129900000000/4.50/16.13/100.48/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=4.11 I.TEMP=14.84 E.PRES=99.01
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/521/31129900000000/4.11/14.84/99.01/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=3.13 I.TEMP=15.38 E.PRES=99.80
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/522/31129900000000/3.13/15.38/99.80/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=4.89 I.TEMP=14.95 E.PRES=97.66
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/523/31129900000000/4.89/14.95/97.66/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=4.60 I.TEMP=15.48 E.PRES=99.80
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/524/31129900000000/4.60/15.48/99.80/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=3.82 I.TEMP=14.84 E.PRES=100.59
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/525/31129900000000/3.82/14.84/100.59/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=2.74 I.TEMP=15.59 E.PRES=100.36
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/526/31129900000000/2.74/15.59/100.36/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=3.72 I.TEMP=14.84 E.PRES=100.03
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/527/31129900000000/3.72/14.84/100.03/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=3.04 I.TEMP=14.52 E.PRES=99.46
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/528/31129900000000/3.04/14.52/99.46/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=2.74 I.TEMP=14.09 E.PRES=99.12
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/529/31129900000000/2.74/14.09/99.12/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=3.72 I.TEMP=14.62 E.PRES=99.24
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/530/31129900000000/3.72/14.62/99.24/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=2.16 I.TEMP=13.76 E.PRES=98.11
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/531/31129900000000/2.16/13.76/98.11/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=1.37 I.TEMP=19.03 E.PRES=96.53
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/532/31129900000000/1.37/19.03/96.53/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=1.08 I.TEMP=22.15 E.PRES=97.32
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/533/31129900000000/1.08/22.15/97.32/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=1.08 I.TEMP=25.81 E.PRES=98.67
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/534/31129900000000/1.08/25.81/98.67/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=1.47 I.TEMP=30.22 E.PRES=97.88
Heat OFF

GET
/RestMissionControl/MissionControl.svc/data/535/31129900000000/1.47/30.22/97.88/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=1.28 I.TEMP=17.42 E.PRES=97.32
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/536/31129900000000/1.28/17.42/97.32/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=0.30 I.TEMP=31.08 E.PRES=97.32
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/537/31129900000000/0.30/31.08/97.32/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-0.09 I.TEMP=36.56 E.PRES=98.00
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/538/31129900000000/-0.09/36.56/98.00/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=0.10 I.TEMP=43.01 E.PRES=96.53
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/539/31129900000000/0.10/43.01/96.53/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-0.39 I.TEMP=29.35 E.PRES=97.43
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/540/31129900000000/-0.39/29.35/97.43/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-1.46 I.TEMP=48.92 E.PRES=97.88
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/541/31129900000000/-1.46/48.92/97.88/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=0.69 I.TEMP=40.65 E.PRES=95.97
Heat OFF
GET
/RestMissionControl/MissionControl.svc/data/542/31129900000000/0.69/40.65/95.97/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-1.66 I.TEMP=31.94 E.PRES=95.63
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/543/31129900000000/-1.66/31.94/95.63/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-1.36 I.TEMP=21.72 E.PRES=94.84
Heat OFF

GET /RestMissionControl/MissionControl.svc/data/544/31129900000000/-1.36/21.72/94.84/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-2.63 I.TEMP=20.22 E.PRES=95.52
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/545/31129900000000/-2.63/20.22/95.52/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-0.78 I.TEMP=33.23 E.PRES=94.84
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/546/31129900000000/-0.78/33.23/94.84/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-3.03 I.TEMP=26.24 E.PRES=95.07
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/547/31129900000000/-3.03/26.24/95.07/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-3.32 I.TEMP=16.34 E.PRES=95.41
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/548/31129900000000/-3.32/16.34/95.41/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-3.51 I.TEMP=15.81 E.PRES=93.94
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/549/31129900000000/-3.51/15.81/93.94/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-3.32 I.TEMP=14.52 E.PRES=94.06
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/550/31129900000000/-3.32/14.52/94.06/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-3.61 I.TEMP=13.55 E.PRES=93.27
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/551/31129900000000/-3.61/13.55/93.27/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-4.49 I.TEMP=13.55 E.PRES=93.49
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/552/31129900000000/-4.49/13.55/93.49/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-4.10 I.TEMP=13.23 E.PRES=92.70
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/553/31129900000000/-4.10/13.23/92.70/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-4.59 I.TEMP=15.05 E.PRES=92.70
Heat OFF

GET /RestMissionControl/MissionControl.svc/data/554/31129900000000/-4.59/15.05/92.70/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-5.57 I.TEMP=17.63 E.PRES=92.37
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/555/31129900000000/-5.57/17.63/92.37/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-5.76 I.TEMP=15.38 E.PRES=92.48
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/556/31129900000000/-5.76/15.38/92.48/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-5.86 I.TEMP=12.15 E.PRES=91.24
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/557/31129900000000/-5.86/12.15/91.24/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-6.64 I.TEMP=11.40 E.PRES=91.24
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/558/31129900000000/-6.64/11.40/91.24/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-6.64 I.TEMP=12.15 E.PRES=91.35
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/559/31129900000000/-6.64/12.15/91.35/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-6.35 I.TEMP=12.15 E.PRES=90.79
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/560/31129900000000/-6.35/12.15/90.79/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-7.23 I.TEMP=11.08 E.PRES=87.07
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/561/31129900000000/-7.23/11.08/87.07/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-6.64 I.TEMP=11.51 E.PRES=91.24
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/562/31129900000000/-6.64/11.51/91.24/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-8.60 I.TEMP=11.08 E.PRES=89.89
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/563/31129900000000/-8.60/11.08/89.89/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-8.01 I.TEMP=21.83 E.PRES=88.09
Heat OFF

GET /RestMissionControl/MissionControl.svc/data/564/31129900000000/-8.01/21.83/88.09/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-8.30 I.TEMP=20.86 E.PRES=89.21
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/565/31129900000000/-8.30/20.86/89.21/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-8.01 I.TEMP=10.43 E.PRES=86.17
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/566/31129900000000/-8.01/10.43/86.17/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-7.72 I.TEMP=9.89 E.PRES=87.97
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/567/31129900000000/-7.72/9.89/87.97/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-8.01 I.TEMP=9.68 E.PRES=87.75
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/568/31129900000000/-8.01/9.68/87.75/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-8.99 I.TEMP=9.03 E.PRES=88.99
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/569/31129900000000/-8.99/9.03/88.99/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-10.16 I.TEMP=9.46 E.PRES=86.96
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/570/31129900000000/-10.16/9.46/86.96/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-9.67 I.TEMP=9.89 E.PRES=87.63
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/571/31129900000000/-9.67/9.89/87.63/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-8.50 I.TEMP=10.86 E.PRES=86.96
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/572/31129900000000/-8.50/10.86/86.96/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-10.65 I.TEMP=9.35 E.PRES=85.94
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/573/31129900000000/-10.65/9.35/85.94/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-11.53 I.TEMP=9.03 E.PRES=87.41
Heat OFF

GET /RestMissionControl/MissionControl.svc/data/574/31129900000000/-11.53/9.03/87.41/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-11.24 I.TEMP=9.14 E.PRES=85.04
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/575/31129900000000/-11.24/9.14/85.04/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-11.33 I.TEMP=8.60 E.PRES=84.26
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/576/31129900000000/-11.33/8.60/84.26/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-12.31 I.TEMP=8.82 E.PRES=83.80
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/577/31129900000000/-12.31/8.82/83.80/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-13.29 I.TEMP=7.85 E.PRES=83.47
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/578/31129900000000/-13.29/7.85/83.47/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-14.07 I.TEMP=8.49 E.PRES=82.12
Heat OFF
GET /RestMissionControl/MissionControl.svc/data/579/31129900000000/-14.07/8.49/82.12/89.559998/44.560001/-100.56/false HTTP/1.1
HOST: 83.44.56.158
E.TEMP=-13.88 I.TEMP=8.60 E.PRES=80.20
Heat OFF
Starting up systems...

The last line is Starting up systems because batteries stopped working.

Annex D. LAUNCH CHECKLIST

- [ ] Inflate the balloon
- [ ] Connect the balloon with the parachute
- [ ] Put the RTD sensor outside the CanSat
- [ ] Connect the feeding system
- [ ] Leave the CanSat open in order to receive GPS signal (5 to 10 minutes)
- [ ] Activate the Vodafone MiFi
- [ ] Once the GPS signal is detected, check if it works with the upper tape
- [ ] Check if the Vodafone MiFi works
- [ ] Check that telemetry is being received
- [ ] Seal the CanSat with the insulation materials
- [ ] Place the RADAR reflector

Annex E. NOTAM REQUEST

GOBIERNO DE ESPAÑA MINISTERIO DE FOMENTO ENAIRE

SOLICITUD DE ACTIVIDAD AÉREA CIVIL RELATIVA A OTROS USOS DEL ESPACIO AÉREO

* A RELLENAR POR EL SOLICITANTE
** A RELLENAR POR EL COP

FECHA*		06/10/16	
REFERENCIA DEL SOLICITANTE*			
REFERENCIA ANTERIOR*		**REFERENCIA ENAIRE****	

1. Solicitante.

Nombre: Yannick Fournier Caiió
Dirección: Calle Arenys, 20 casa 3, 08860, Castelldefels, Barcelona
Teléfono: 936365298
Móvil: 626650957
Correo Electrónico: yannickfc94@hotmail.com

2. Naturaleza de la actividad.

- ☐ Láser / Focos.
- ☐ Fuegos Artificiales.
- ☐ Suelta de Farolillos.
- ☐ Suelta de Globos
- ☐ Sondeos Meteorológicos.
- ☐ Globos Cautivos.
  - ☐ Publicidad.
  - ☐ Pasajeros.
  - ☐ Fotografía y Filmación.
- ☐ Área Segregada Temporal.
  - ☐ Publicada en el AIP (identificación y nombre): .....
  - ☐ Por Motivos de Seguridad para Acontecimiento Público o Privado: .....
- ☒ Otros: Ascenso satélite tipo CANSAT (no orbital) de 700gr mediante globo aerostático

3. Declaración de Autorización.

Por la presente designo y autorizo a Jordi Gutiérrez Cabello – UPC (EETAC) a actuar como representante en la tramitación de este formulario de solicitud de permiso, y coordinador de la actividad aeronáutica a realizar, pudiendo aportar, si se requiere, la información suplementaria necesaria.

Firma del Organizador

4. Fechas de la Actividad.

Fechas: Sábado 29 de Octubre de 2016
Horarios (Indicar si es Hora Local o UTC): 09:00 (Hora Local)
Duración de la Actividad: 4 horas

5. Zona de Trabajo y Características de la Actividad (sistema de referencia WGS-84. Coordenadas geográficas).

Municipio y Provincia: Mollerussa, Lérida

A. Tipo de Zona (en grados, minutos y segundos. Indicar longitud este u oeste. Añadir tantos puntos como sea necesario).

☐ Área circular / ☒ Punto / ☐ Polígono / ☐ Trayectoria.

Latitud	41°36'41"	Longitud	0°51'18"	☐ W/☒ E	Latitud		Longitud		☐ W/☐ E
Latitud		Longitud		☐ W/☐ E	Latitud		Longitud		☐ W/☐ E
Latitud		Longitud		☐ W/☐ E	Latitud		Longitud		☐ W/☐ E
Latitud		Longitud		☐ W/☐ E	Latitud		Longitud		☐ W/☐ E
Latitud		Longitud		☐ W/☐ E	Latitud		Longitud		☐ W/☐ E

Radio 10 ☒ Km / ☐ Nm / ☐ m

B. Altura sobre el Terreno (AGL) o Altitud sobre el Nivel del Mar (AMSL).

☒ m ☐ ft / ☐ AGL ☒ AMSL

Indicar unidad y tipo: 35000

C. Otros Datos:

6. Características de la Actividad (información adicional para las siguientes actividades).

☒ **A. Sondeos Meteorológicos.**

Tipo de sondeo: ☒ Ligero / ☐ Medio / ☐ Pesado.

Diámetro máximo del globo:	3	m	Peso de la sonda / globo:	0.7/1.2	Kg
Color del globo:	Blanco		Número de globos:	1	
Régimen de ascenso:	4,7	m/sg	Régimen de descenso:	3-33	m/sg

☐ **B. Suelta de Farolillos / Suelta de Globos.**

Número de globos: _____ Diámetro: _____ Color: _____

☐ **C. Láser / Foco.**

Barrido horizontal del haz (Entre 0º y 360º):

Barrido vertical del haz (Entre la horizontal 0º y 90º):

☒ **D. Otros Datos:** Actividad científica de adquisición y procesamiento de datos atmosféricos

7. Declaración de Conformidad.

Declaro que:

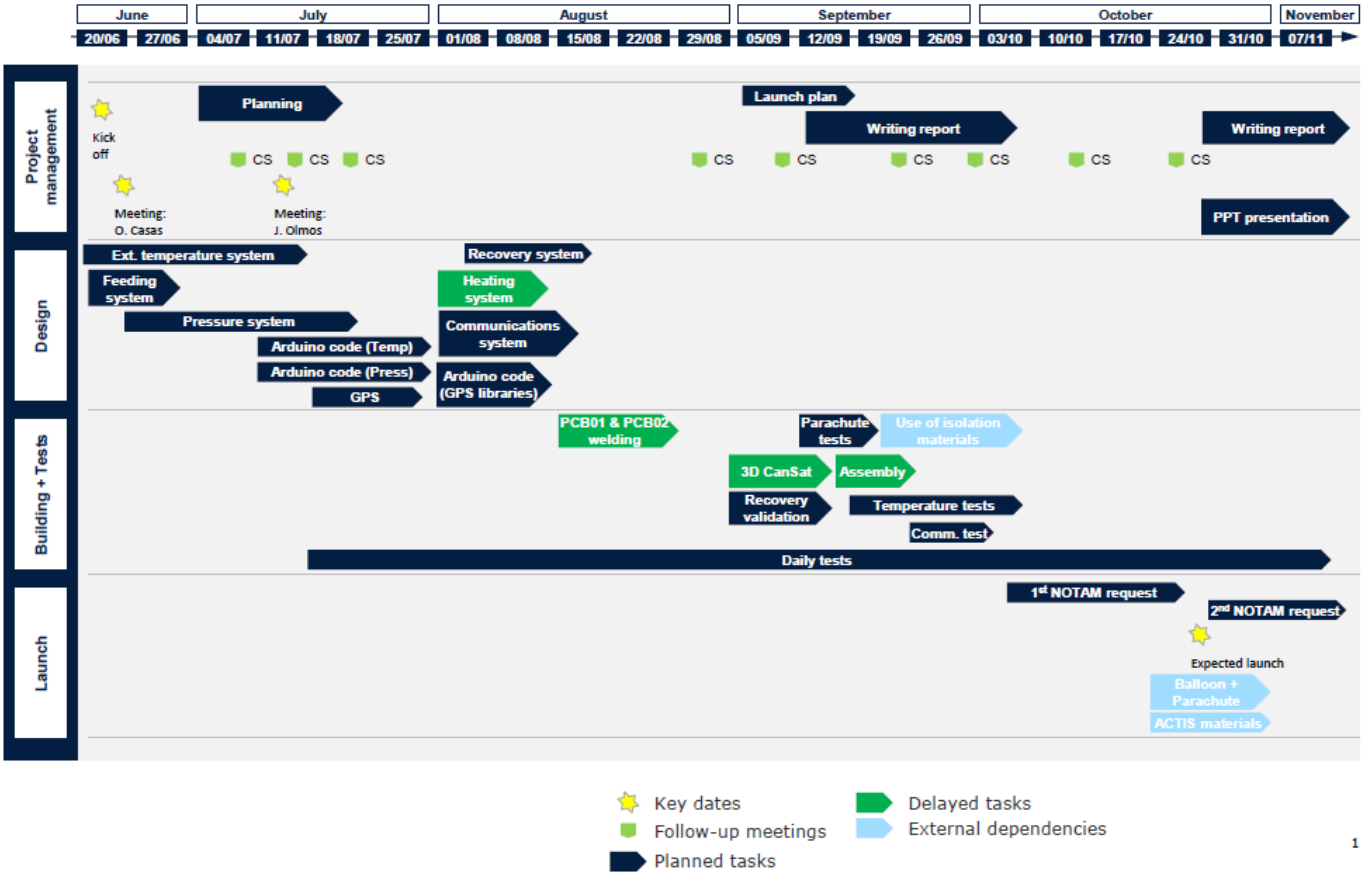
1. La información contenida en este formulario, así como la documentación adjunta, es real, verdadera y correcta.
2. Cuento con la habilitación necesaria para poder acometer la actividad solicitada.
3. El personal y/o medios materiales empleados para realizar la actividad cumple con la normativa vigente así como con los requisitos establecidos por la DGAC (Dirección General de Aviación Civil) / AESA (Agencia Estatal de Seguridad Aérea).

Firma

Remitir a:
ENAIRE
Dirección de Operaciones / GCAT
Dpto. Coordinación Operativa del Espacio Aéreo (COP)
Avda. de Aragón 402
Edificio Lamela, 4ª Planta
28022 Madrid
Teléfono: 913 213 378
Correo Electrónico: cop@enaire.es

Annex F. GANTT DIAGRAM AND HOUR JOURNAL

Gantt diagram



Hour journal

MES	DIA	ACTIVIDAD	HORAS		TRABAJO AUTÓNOMO
JUNIO	15	Ruenión inicial para planificación proyecto CANSAT	3	YANNICK	54
	20	Reunión Oscar Casas (Alimentación)	1	DAVID	54
JULIO	5	Reunión de proyecto 1	1		
	8	Diseño sistema de temperatura (laboratorio)	3		
	12	Reunión de proyecto 2	1		
	13	Diseño sistema de presión (laboratorio)	2		
	14	Reunión Joan Olmos (GPS, XBEE, Recuperación)	2		
	15	Diseño y montaje sistema de temperatura (laboratorio)	3		
	19	Reunión de proyecto 3	1		
	20	Diseño y prueba sistema GPS	3		
	21	Validar sistema de presión del 19/07	3		
	25	Montaje de pres + temp en diferentes placas Protoboard	1		
	26	Reunión en serveis técnics para la elaboracón de placa impresa	1		
AGOSTO	3	Inicio diseño comunicaciones XBEE	2		
	4	Inicio diseño sistema de recuperación	2		
	9	Hablar de la alimentación, aislantes térmicos y climatización	2		
	10	Diseño en placa de los sistemas de temperatura y presión	3		
	11	Probar XBEE y ver que las comunicaciones funcionan	5		
	12	Diseñar sistema de recuperación	4		
	16	Soldadura sistema de temperatura	4		
	17	Soldadura sistema de presión	4		
	18	Código temperatura y presión	4		
	19	Compra componentes (pilas, conectores, etc..)	2		
	22	Soldadura sistema climatización y recuperación	2		
	23	Lab. Soldadura climatización y recuperación + consolidar Placas Temp. y Pres.	4		
	24	Diseño de la carcasa + consolidación placas soldadas	5		
	25	Primera prueba de comunicaciones (universidad)	3		
	29	Reunión para organización redactión	3		
SEPTIEMBR	3	Código sistema de recuperación y primera validación	11		

	4	Validación sistema de recuperación	2		
	10	Leroy Merlin+Diotronics+Paracaida+Reflector+Blister Pilas	9		
	11	Servidor+Tornillos+Prueba global	7		
	14	Prueba de paracaídas con objeto de 700gr	3		
	15	Empezar con el ensamblaje en CANSAT	3		
	16	Terminar el ensamblaje del CANSAT y comprobar funcionamiento	3		
	17	Primera prueba temperatura congelador + mejora de código	5		
	20	Segunda prueba temperatura congelador	4		
	21	Prueba de funcionamiento + tercera prueba de temperatura congelador	4		
	22	Gancho superior + cálculo comunicaciones	2		
	24	Prueba de comunicaciones (Montserrat - Collserola)	4		
	27	Mejoras en código	3		
	30	Planificación de la redacción	2		
OCTUBRE	4	Cuarta prueba de temperatura	4		
	7	Unión de los 3 primeros capítulos y últimas pruebas	4		
	20	Compra globo y paracaídas	2		
	23	Planificación de las próximas acciones	2		
	24	Visita a ACTIS (aislantes térmicos) para colaboración	2		
	26	Recubrimiento aislante + sensor temperature externa (agujeros)	3		
NOVIEMBRE	1	Inicio presentación PPT	3		
	2	Reunión planificación última semana	2		
	3	Redacción últimos capítulos	3		
	4	Redacción últimos capítulos	2		
	5	Revisión ortográfica de las memorias	8		
	6	Dar formato a las memorias	6		
		TOTAL	177		
		TOTAL + Autonomo	285		

Annex G. MATERIALS AND COST SHEET

Product	Quantity	Cost
Energizer 522 Battery, Alkaline, 625mAh, 9V	24	26,45 €
Bud Industries HH-3449 Battery Retainer Clip	3	3,99 €
Arduino UNO ATmega328 Eval Board	1	29,39 €
Texas Instruments Temperature Sensor, LM35DT, TO-220	1	5,76 €
Digi International XBee PRO 868	2	172,16 €
Linx Technologies Inc. ANT-868-CW-RAH-XXX	1	7,88 €
ALSrobot XBee USB Explorer Dongle	1	10,64 €
Arduino XBee Shield	1	18,65 €
U-blob GPS Module (With Antenna)	1	17,15 €
Pt-1000 RTD, 1kohm, -200ºC-600ºC	4	19,43 €
MAX 660 CMOS voltage inverter	1	11,00 €
MPX2200AP Sensor de Presión, Absoluta, 0.2 mV/kPa, 0 kPa, 200 kPa, 10 V, 16 V	1	13,35 €
Actis thermal insulation tape	1	8,90 €
Dupont cables	1	6,49 €
Aluminium thermal insulation	1	25,90 €
SMA antenna adapter for XBee	2	5,07 €
Construction material (screws and tools)	1	14,44 €
Parachute 1 (Umbrella)	1	5,65 €
3.3V Voltage Regulators	3	1,29 €
Electronic components (Resistors, capacitors and operational amplifiers)	1	35,40 €
Energizer 522 Lithium Battery, 800 mAh, 9V	4	38,84 €
Modem MiFi	1	38,69 €
Soldering tin 100gr	1	7,95 €
ESP8266 Wi-FI Module for Arduino	3	23,70 €
Stratoflights Weather Balloon 1600	1	195,99 €
Stratoflight Parachute 800	1	28,65 €
Helium 9m3 bottle	1	200,00 €
Vodafone 1GB SIM card	1	10,00 €
Total (includes taxes)		**982,80 €**

Annex H. RELATED EETAC COURSES

COURSE	CONCEPT
Aerospace Technology and Air Transportation	Parachute lift calculations
Thermodynamics	Heat transfer and thermal aisle.
Informatics II	Server side programming in C# for the Power Bi dashboard.
Air Transportation Infrastructures	Air space classification, election of the launching place, meteorological considerations and administrative procedures (NOTAM).
Electricity	Power consumption design (chose of resistors)
Science and Technology of Materials	Election of the aluminum composite for the thermal aisle and study about the mechanical properties of the plastic of the structure.
Linear Systems	Filter design.
Graphical Expression	3D Design of the structure.
Fundamentals of Communication	Election of communication devices and operative frequencies.
Electronics	Design of the electronics needed to read the information from the sensors. Use of a microcontroller.
Flight Operations	Administrative work and order of NOTAM.
Aeronautical Communications	Computations of signal quality parameters for the communications. Estimation of noise.
Avionics	Chose of the appropriate sensor type and electronic calculations (such as dynamic range)
Design and Test of Aeronautical Projects	Validations and design considerations (thermal design, voltage range accommodation, power supply system).
CES	Signal link budget equations. Study of signal attenuations.
NACC	Navigation concepts, GPS knowledge.